



Master's Thesis

For the Award of the Master's Degree

Field of Study: Electronics  
Specialization: Industrial Computing

Presented by:

Abou Baker GUEBRILI

Ines TRARI MEDJAOUI

Thesis Title

**Intelligent Hydroponic Crop Management System  
Using Artificial Intelligence, Digital Twin  
Technology,  
Fuzzy Logic Control, and Chatbot Assistance**

Publicly defended on June 18, 2026, before the Examination Committee composed of:

|                            |           |                                               |                          |
|----------------------------|-----------|-----------------------------------------------|--------------------------|
| Ms. Nadjia MIDOUN          | MCB       | ESSAT                                         | Chair                    |
| Mr. Ghouti ABDELLAOUI      | MCA       | ESSAT                                         | Supervisor               |
| Ms. Amina OUHOUD           | MCA       | ESSAT                                         | Supervisor               |
| Mr. Chamseddine BEMMOUSSET | MCA       | Ministry of Knowledge<br>Economy and Startups | Co-Supervisor            |
| Ms. Mokhtaria BEKKAOU      | MCB       | ESSAT                                         | Examiner                 |
| Ms. Houda DERBAL           | Director  | ONID                                          | Socio-Economic Partner   |
| Ms. Wassila TABET AOUEL    | Professor | University of Tlemcen                         | Incubator Representative |

Academic Year: 2025–2026

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

# Dedication

*I dedicate this thesis first and foremost to my beloved parents, **Reda** and **Souhila**, who taught me the value of perseverance and encouraged me to pursue my goals with determination. Their unwavering moral support, sacrifices, and financial assistance have played a fundamental role in shaping the person I am today.*

*I would also like to dedicate this work to my dear sisters, each of whom contributed to my journey in her own unique way: **Nihel**, for her strong sense of responsibility; **Razane**, for her attentive listening and support; **Ikram**, for her wisdom and guidance; and **Manel**, for her positive energy and cheerful spirit.*

*My heartfelt gratitude goes to my fiancé, **Salah Eddine**, whose constant support, encouragement, and understanding helped me overcome many challenges throughout this journey. I would also like to thank my future in-laws for their kindness and support.*

*I extend my appreciation to my entire family for their encouragement and belief in me.*

*Finally, I dedicate this work to my students, whose patience, trust, and continuous encouragement inspired me to keep moving forward and gave me the motivation to achieve this important milestone.*

*To all of you, thank you for being part of my journey and for contributing to this achievement.*

# Dedication

*To my parents,*

*There are no words in any language vast enough to hold what you mean to me. You carried my dreams before I even knew I had them. every word of faith you whispered when I doubted myself — it is all woven into every page of this work. This thesis is not mine alone; it is the fruit of everything you planted in me. I owe you not a dedication, but a lifetime.*

*To my two brothers And to my two little sisters, my greatest sources of joy and tenderness — watching you grow has reminded me, every single day, why it is worth giving your best.*

— ★ —

*To Malak BRIXI,*

*There is a particular kind of strength that does not announce itself — it simply shows up, quietly, every time you need it. That is what you have been to me. Through the exhaustion, the doubt, and the long stretches of uncertainty, you never let me walk alone. Your encouragement was not just kindness; it was fuel when I had nothing left. Thank you for believing in this journey even on the days I struggled to believe in it myself. Some people pass through your story. You became part of it.*

— ★ —

*To my brothers of five years,*

*Five years. That is not just time — that is a shared life. Sid Ahmed, Ahmed, Habib, Aissam, Amin, Zinou, Salah Eddine — we built something rare together. We pushed through the hardest exams, laughed through the absurd hours, and carried each other through everything in between. The memory of this journey is inseparable from your faces, your voices, your presence.*

*And to Abdelrahman RAZALI and Mouadh ABDELLAOUI — you are more than friends. Through every challenge, every deadline, and every moment of doubt, your support and friendship never failed. This journey would not have been the same without you. Brothers in every sense that truly matters.*



*To every member of TexSquad,*

*To the team that chose ambition over comfort and collaboration over competition — you know what we built, and you know what it cost. What we created together goes far beyond any project or deliverable. We created something we can be proud of. Always.*

*“And He found you lost and guided you.”*

*— Quran, Ad-Duha: 7*

# Acknowledgement

We would like to express our deepest gratitude to our supervisors, **Ms. Amina Ouhoud**, **Mr. Chems-Eddine Benmoussat**, and **Mr. Ghouti Abdellaoui**, for their invaluable guidance throughout the development of this work. Their dedication, insightful advice, and continuous support have been instrumental in shaping this project and greatly enriched our academic journey. We would like to extend a special recognition to **Mr. Ghouti Abdellaoui**, who served not only as our supervisor but also as the Head of Specialty. His vision and initiative in creating this new specialty — of which we are proud to be the very first graduating class — have been foundational to our academic path, and we are deeply grateful for the opportunity he gave us.

We extend our sincere thanks to **Mr. Ramzedinne Adjim**, Head of the FabLab, whose technical expertise and generous assistance proved crucial to the realization of our prototype. His hands-on involvement and practical guidance were indispensable at every stage of the hardware development.

Our gratitude also goes to the entire staff of the Higher School of Applied Sciences Tlemcen, whose collective efforts contributed to the successful completion of this project.

Finally, we would like to thank **Mr. Boucherit Rouissat**, Director of ESSA Tlemcen, and **Mr. Sidi-Mohamed Khefif** for granting us the opportunity to work within the FabLab facilities, providing us with the resources and environment necessary to bring this project to life.

## Abstract

This thesis presents the design and implementation of an intelligent hydroponic system combining artificial intelligence and IoT technologies. The first chapter establishes the theoretical background of the project. The second chapter describes two AI models: a plant disease detection model built upon three architectures (Custom CNN, EfficientNet-B3, and ResNet-50), evaluated on 20 disease classes covering tomato, lettuce, and strawberry plants, and a crop yield estimation model based on YOLO. Both models achieved strong performance. The third chapter presents the system's digital twin, integrating a Mamdani fuzzy logic controller for automatic nutrient and pH regulation, along with an embedded conversational assistant, HydroBot, enabling intuitive system monitoring and interaction.

**Keywords:** Hydroponics, Deep Learning, CNN, EfficientNet-B3, ResNet-50, YOLO, Disease Detection, Digital Twin, Fuzzy Logic, IoT, Chatbot.

## Résumé

Ce mémoire présente la conception et l'implémentation d'un système hydroponique intelligent combinant intelligence artificielle et technologies IoT. Le premier chapitre pose les bases théoriques du projet. Le deuxième chapitre décrit deux modèles d'IA développés : un modèle de détection des maladies des plantes basé sur trois architectures (CNN personnalisé, EfficientNet-B3 et ResNet-50), évalué sur 20 classes de maladies affectant la tomate, la laitue et la fraise, ainsi qu'un modèle d'estimation du poids de la récolte reposant sur YOLO. Les deux modèles ont produit de bons résultats. Le troisième chapitre présente le jumeau numérique du système, intégrant une logique floue de type Mamdani pour le contrôle automatique des nutriments et du pH, ainsi qu'un chatbot embarqué, HydroBot, facilitant la supervision et l'interaction avec le système.

**Mots-clés :** Hydroponique, Deep Learning, CNN, EfficientNet-B3, ResNet-50, YOLO, Détection de maladies, Jumeau numérique, Logique floue, IoT, Chatbot.

## الملخص

يُقدّم هذا المذكرة تصميم وتنفيذ نظام زراعة مائية ذكي يجمع بين الذكاء الاصطناعي وتقنيات إنترنت الأشياء. يضع الفصل الأول الأسس النظرية للمشروع، فيما يصف الفصل الثاني نموذجين للذكاء الاصطناعي : الأول مخصص للكشف عن أمراض النباتات باستخدام ثلاث بنىات معمارية (شبكة ث مخصصة، ويففخ-نت ٣٠٥، وغس ٥٠٥)، وتم تقييمه على ٢٠ صنفًا من الأمراض التي تصيب الطماطم والخس والفراولة، والثاني مخصص لتقدير وزن المحصول باستخدام نموذج بوبو. وقد أسفر كلا النموذجين عن نتائج جيدة. أما الفصل الثالث فيتناول التوأم الرقمي للنظام، الذي يدمج منطق ضبابيًا من نوع بدمن للتحكم التلقائي في العناصر الغذائية ومستوى الحموضة، إضافةً إلى مساعد محادثة مدمج يُسمى حيدر-٣٠٥ يتيح مراقبة النظام والتفاعل معه بشكل سهل وبديهي.

**الكلمات المفتاحية ~ :** الزراعة المائية، التعلم العميق، ث ، يففخ-نت ٣٠٥، غس ٥٠٥، بوبو، كشف الأمراض، التوأم الرقمي، المنطق الضبابي، إنترنت الأشياء، روبوت المحادثة.

# Contents

|                                                                                      |           |
|--------------------------------------------------------------------------------------|-----------|
| <b>List of Figures</b>                                                               | <b>10</b> |
| <b>List of Tables</b>                                                                | <b>12</b> |
| <b>Abbreviations list</b>                                                            | <b>12</b> |
| <b>Introduction</b>                                                                  | <b>13</b> |
| <b>1 Fundamentals of Artificial Intelligence for Smart Agriculture</b>               | <b>15</b> |
| Introduction . . . . .                                                               | 15        |
| <b>Part 1: Artificial Intelligence</b>                                               | <b>17</b> |
| 1 Machine Learning . . . . .                                                         | 17        |
| 1.1 Supervised Learning . . . . .                                                    | 17        |
| 1.2 Unsupervised Learning . . . . .                                                  | 17        |
| 1.3 Reinforcement Learning . . . . .                                                 | 17        |
| 2 Deep Learning . . . . .                                                            | 17        |
| 2.1 Neural network vocabulary . . . . .                                              | 18        |
| 3 CNN architecture . . . . .                                                         | 20        |
| 3.1 Transfer Learning . . . . .                                                      | 25        |
| 4 ResNet . . . . .                                                                   | 26        |
| 4.1 EfficientNet-B3 . . . . .                                                        | 26        |
| 5 Object detection and Segmentation . . . . .                                        | 27        |
| 5.1 What is Yolo? . . . . .                                                          | 28        |
| 5.2 Evolution of YOLO Models . . . . .                                               | 28        |
| 5.3 YOLO 11 Architecture . . . . .                                                   | 29        |
| 6 Evaluation Metrics . . . . .                                                       | 31        |
| 6.1 Classification Metrics . . . . .                                                 | 32        |
| 6.2 Object Detection Metrics . . . . .                                               | 33        |
| 7 State of the Art of Artificial Intelligence in Agriculture and Hydroponics . . . . | 34        |
| <b>Part 2: Fuzzy Logic</b>                                                           | <b>41</b> |
| 7.1 Definition and Basic Concepts . . . . .                                          | 41        |
| 7.2 Difference Between Classical Logic and Fuzzy Logic . . . . .                     | 42        |

|          |                                                                                                          |           |
|----------|----------------------------------------------------------------------------------------------------------|-----------|
| <b>2</b> | <b>Design and Modeling of Proposed AI Solutions</b>                                                      | <b>43</b> |
|          | Introduction . . . . .                                                                                   | 43        |
| 1        | Plant Disease Detection Models . . . . .                                                                 | 43        |
| 1.1      | Dataset Acquisition and Processing . . . . .                                                             | 43        |
| 1.2      | Rationale for Architecture Selection . . . . .                                                           | 47        |
| 1.3      | Training Environment . . . . .                                                                           | 47        |
| 1.4      | Custom Convolutional Neural Network . . . . .                                                            | 48        |
| 1.5      | ResNet-50 . . . . .                                                                                      | 49        |
| 1.6      | EfficientNet-B3 . . . . .                                                                                | 50        |
| 1.7      | Summary of Hyperparameters . . . . .                                                                     | 50        |
| 1.8      | CNN Results . . . . .                                                                                    | 51        |
| 1.9      | ResNet-50 Results . . . . .                                                                              | 52        |
| 1.10     | EfficientNet-B3 Results . . . . .                                                                        | 54        |
| 1.11     | Three-Model Comparison . . . . .                                                                         | 56        |
| 2        | Tomato Instance Segmentation Using YOLO11 . . . . .                                                      | 59        |
| 2.1      | Dataset Description . . . . .                                                                            | 60        |
| 2.2      | Training Configuration . . . . .                                                                         | 61        |
| 2.3      | Experimental Results . . . . .                                                                           | 62        |
| 2.4      | Class-wise Performance . . . . .                                                                         | 66        |
| 2.5      | Tomato Yield Estimation . . . . .                                                                        | 66        |
| 2.6      | Conclusion . . . . .                                                                                     | 67        |
| <b>3</b> | <b>Intelligent Hydroponic System: Fuzzy Logic Control, Digital Twin Modelling, and Conversational AI</b> | <b>69</b> |
|          | Introduction . . . . .                                                                                   | 69        |
| 1        | Digital Twin Integration in the VertiFarm Platform . . . . .                                             | 69        |
| 1.1      | Concept and Definition of Digital Twins . . . . .                                                        | 69        |
| 1.2      | Digital Twins in Smart Agriculture . . . . .                                                             | 70        |
| 1.3      | The VertiFarm Digital Twin Architecture . . . . .                                                        | 70        |
| 1.4      | Digital Twin Functional Layers . . . . .                                                                 | 72        |
| 1.5      | Real-Time State Synchronisation . . . . .                                                                | 72        |
| 1.6      | Behavioural Modelling — Fuzzy Logic Controller . . . . .                                                 | 73        |
| 1.7      | Anomaly Detection — AI Disease Identification . . . . .                                                  | 80        |
| 1.8      | Operator Interface — Dashboard as Digital Twin Window . . . . .                                          | 81        |
| 1.9      | Related Work on Digital Twins in Hydroponics . . . . .                                                   | 81        |
| 1.10     | Comparison of VertiFarm with Related Digital Twin Works . . . . .                                        | 82        |
| 2        | HydroBot: A Domain-Constrained Conversational Assistant for the Hydroponic System . . . . .              | 83        |
| 2.1      | Motivation and Design Objectives . . . . .                                                               | 83        |
| 2.2      | System Architecture . . . . .                                                                            | 83        |
| 2.3      | Domain Filtering and Safety Mechanism . . . . .                                                          | 84        |
| 2.4      | System Prompt Engineering . . . . .                                                                      | 86        |
| 2.5      | Offline LLM Inference with Ollama . . . . .                                                              | 86        |
| 2.6      | User Interface and Operator Interaction . . . . .                                                        | 87        |
|          | <b>General Conclusion</b>                                                                                | <b>90</b> |

# List of Figures

|      |                                                                                                                           |    |
|------|---------------------------------------------------------------------------------------------------------------------------|----|
| 1.1  | Cost function principle based on the distance between prediction and target .                                             | 18 |
| 1.2  | Gradient descent principle for minimizing the cost function . . . . .                                                     | 18 |
| 1.3  | Learning rate . . . . .                                                                                                   | 19 |
| 1.4  | Backpropagation . . . . .                                                                                                 | 19 |
| 1.5  | Example of epochs . . . . .                                                                                               | 20 |
| 1.6  | CNN architecture . . . . .                                                                                                | 21 |
| 1.7  | Fully Connected Layer . . . . .                                                                                           | 21 |
| 1.8  | Convolution process in a CNN using a 3×3 kernel to extract image features .                                               | 23 |
| 1.9  | Principle of max pooling in a CNN. . . . .                                                                                | 24 |
| 1.10 | Comparison between max pooling and average pooling . . . . .                                                              | 25 |
| 1.11 | ResNet architecture based on convolutional and identity blocks with skip connections . . . . .                            | 26 |
| 1.12 | EfficientNet architecture based on MBConv blocks . . . . .                                                                | 27 |
| 1.13 | Comparison between image classification, object detection, and image segmentation . . . . .                               | 28 |
| 1.14 | Architecture of YOLOv11 [1] . . . . .                                                                                     | 30 |
| 1.15 | Common evaluation metrics used for classification and regression tasks [2] .                                              | 32 |
| 1.16 | Confusion matrix for classification performance evaluation [3] . . . . .                                                  | 33 |
| 1.17 | Evaluation metrics for object detection using IoU and mAP . . . . .                                                       | 34 |
| 1.18 | Distribution of reviewed studies by publication year and research domain. . .                                             | 39 |
| 1.19 | fuzzy logic . . . . .                                                                                                     | 41 |
| 2.1  | Sample images from the dataset illustrating various plant disease classes across tomato, lettuce, and strawberry. . . . . | 45 |
| 2.2  | Integration pipeline architecture: from raw source datasets to the final stratified train/val/test split. . . . .         | 48 |
| 2.3  | CNN training history over 50 epochs. . . . .                                                                              | 51 |
| 2.4  | CNN test set evaluation. . . . .                                                                                          | 51 |
| 2.5  | ResNet-50 training history over 20 epochs. The vertical dashed line marks the Phase A to Phase B transition. . . . .      | 52 |
| 2.6  | ResNet-50 test set evaluation. . . . .                                                                                    | 53 |
| 2.7  | EfficientNet-B3 training history over 20 epochs . . . . .                                                                 | 54 |
| 2.8  | EfficientNet-B3 confusion matrix and per-class $F_1$ scores on the test set. . . .                                        | 55 |
| 2.9  | <i>Strawberry</i> — angular leafspot. <b>Predicted:</b> angular leafspot. . . . .                                         | 58 |
| 2.10 | <i>Lettuce</i> — fungal. <b>Predicted:</b> fungal. . . . .                                                                | 58 |
| 2.11 | <i>Tomato</i> — spider mites. <b>Predicted:</b> spider mites. . . . .                                                     | 59 |
| 2.12 | Yolo V11 Instance segmentation Architecture . . . . .                                                                     | 60 |
| 2.13 | Examples of the three tomato ripeness categories from the Laboro Tomato Dataset. . . . .                                  | 61 |

|      |                                                                                                         |    |
|------|---------------------------------------------------------------------------------------------------------|----|
| 2.14 | Bounding Box F1-Confidence Curve for YOLO11s-seg . . . . .                                              | 62 |
| 2.15 | Bounding Box Recall-Confidence Curve for YOLO11s-seg . . . . .                                          | 63 |
| 2.16 | Bounding Box Precision-Recall Curve for YOLO11s-seg . . . . .                                           | 64 |
| 2.17 | Confusion Matrix for YOLO11s-seg on the Validation Set . . . . .                                        | 65 |
| 2.18 | Tomato yield estimation: (left) raw input image; (right) YOLO11s-seg . . . .                            | 67 |
| 3.1  | General Digital Twin architecture . . . . .                                                             | 70 |
| 3.2  | Layered architecture of the VertiFarm Digital Twin. . . . .                                             | 72 |
| 3.3  | Fuzzy logic control flow of the VertiFarm system from sensor acquisition to actuator execution. . . . . | 73 |
| 3.4  | Automatic dosing mode using the fuzzy logic controller. . . . .                                         | 74 |
| 3.5  | Manual dosing mode allowing direct operator control of valves. . . . .                                  | 75 |
| 3.6  | End-to-end architecture of the local AI chatbot system integrated into the VertiFarm platform. . . . .  | 84 |
| 3.7  | Blocked-topic detection in HydroBot. . . . .                                                            | 85 |
| 3.8  | HydroBot chatbot interface integrated into the VertiFarm dashboard. . . . .                             | 88 |

# List of Tables

|      |                                                                                                                                                                                                                                                                                                                         |    |
|------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 1.1  | Hyperparameters according to the classification problem . . . . .                                                                                                                                                                                                                                                       | 22 |
| 1.2  | YOLO: Evolution of models . . . . .                                                                                                                                                                                                                                                                                     | 29 |
| 1.3  | Main classification metrics derived from the confusion matrix . . . . .                                                                                                                                                                                                                                                 | 33 |
| 1.4  | Comparison between Classical Logic and Fuzzy Logic . . . . .                                                                                                                                                                                                                                                            | 42 |
| 2.1  | Summary of the three source datasets prior to preprocessing. . . . .                                                                                                                                                                                                                                                    | 44 |
| 2.2  | Augmentation pipeline applied to minority classes, with biological and technical justification. . . . .                                                                                                                                                                                                                 | 46 |
| 2.3  | Preprocessing and online augmentation pipeline. . . . .                                                                                                                                                                                                                                                                 | 46 |
| 2.4  | Final dataset distribution after augmentation and stratified 80/10/10 split. . .                                                                                                                                                                                                                                        | 47 |
| 2.5  | Architectural properties of the three evaluated models. . . . .                                                                                                                                                                                                                                                         | 47 |
| 2.6  | Custom CNN architecture — layer sequence and output feature map dimensions (input: $3 \times 224 \times 224$ ). . . . .                                                                                                                                                                                                 | 49 |
| 2.7  | Training hyperparameters for all three models. . . . .                                                                                                                                                                                                                                                                  | 50 |
| 2.8  | CNN per-class precision, recall, and $F_1$ on the test set (2,326 images, 20 classes). .                                                                                                                                                                                                                                | 52 |
| 2.9  | ResNet-50 per-class precision, recall, and $F_1$ on the test set (2,326 images, 20 classes). . . . .                                                                                                                                                                                                                    | 54 |
| 2.10 | EfficientNet-B3 per-class precision, recall, and $F_1$ on the test set (2,326 images, 20 classes). . . . .                                                                                                                                                                                                              | 56 |
| 2.11 | Three-model comparison — performance, efficiency, and training metrics. . .                                                                                                                                                                                                                                             | 57 |
| 2.12 | YOLO11s-seg training configuration. . . . .                                                                                                                                                                                                                                                                             | 61 |
| 2.13 | Overall Model Performance . . . . .                                                                                                                                                                                                                                                                                     | 62 |
| 2.14 | Segmentation Performance per Class . . . . .                                                                                                                                                                                                                                                                            | 66 |
| 3.1  | Physical entity instrumentation of the VertiFarm Digital Twin. . . . .                                                                                                                                                                                                                                                  | 71 |
| 3.2  | pH input membership function parameters for all three crop profiles. $\text{zmf}(a, b)$ = Z-shaped function (full membership below $a$ , zero at $b$ ); $\text{trimf}(a, b, c)$ = triangular (peak at $b$ ); $\text{smf}(a, b)$ = S-shaped (zero at $a$ , full membership above $b$ ). All values are pH units. . . . . | 76 |
| 3.3  | EC input membership function parameters for all three crop profiles. Function types are identical to Table 3.2. All values are in $\text{mS}\cdot\text{cm}^{-1}$ . . . . .                                                                                                                                              | 76 |
| 3.4  | Output membership functions for valve dosing time (seconds). All six levels use triangular $\text{trimf}(a, b, c)$ functions except <i>none</i> (left-edge, implemented as $\text{zmf}$ ) and <i>max</i> (right-edge, implemented as $\text{smf}$ ). Volume computed at flow rate = 83 mL/s. . . . .                    | 77 |
| 3.5  | Fuzzy rule base implemented in <code>evaluate_rules()</code> — logical conditions, activated output levels, and indicative centroid dose values (Lettuce profile, 83 mL/s flow rate). . . . .                                                                                                                           | 78 |
| 3.6  | Fuzzy inference engine validation results (Lettuce profile). . . . .                                                                                                                                                                                                                                                    | 80 |

|     |                                                                                                             |    |
|-----|-------------------------------------------------------------------------------------------------------------|----|
| 3.7 | Comparative analysis of VertiFarm against related hydroponic Digital Twin implementations. . . . .          | 82 |
| 3.8 | Evaluation of local LLM candidates for Raspberry Pi 5 deployment under the VertiFarm memory budget. . . . . | 87 |
| 3.9 | Representative operator queries and HydroBot response scope. . . . .                                        | 88 |

# Abbreviations list

| Acronym | Signification                                    |
|---------|--------------------------------------------------|
| AI      | Artificial Intelligence                          |
| AP      | Average Precision                                |
| BN      | Batch Normalization                              |
| CE      | Cross-Entropy                                    |
| CNN     | Convolutional Neural Network                     |
| CSV     | Comma-Separated Values                           |
| DFL     | Distribution Focal Loss                          |
| EDA     | Exploratory Data Analysis                        |
| F1      | F1-Score (harmonic mean of Precision and Recall) |
| GAP     | Global Average Pooling                           |
| GPU     | Graphics Processing Unit                         |
| IoU     | Intersection over Union                          |
| JSON    | JavaScript Object Notation                       |
| LR      | Learning Rate                                    |
| mAP     | mean Average Precision                           |
| PR      | Precision-Recall                                 |
| ReLU    | Rectified Linear Unit                            |
| VRAM    | Video Random Access Memory                       |
| YOLO    | You Only Look Once                               |

# Introduction

Agriculture is undergoing a profound transformation driven by the convergence of artificial intelligence, embedded systems, and advanced cultivation techniques. As global food demand continues to rise alongside growing environmental constraints, traditional farming methods are increasingly inadequate to ensure sustainable and efficient crop production. In this context, smart agriculture emerges as a compelling paradigm, combining data-driven decision-making with automated and connected systems to optimize plant growth, detect diseases early, and minimize resource waste.

This work addresses the design and implementation of an intelligent, modular, and connected hydroponic system enhanced by artificial intelligence and digital twin technology. Hydroponics, as a soil-free cultivation method, offers precise control over nutrient delivery and growth conditions, making it particularly well-suited for integration with automated monitoring and AI-based diagnostics. By embedding intelligence directly into the cultivation infrastructure, it becomes possible to move from reactive to predictive and prescient agricultural management.

This memoir is organized around three complementary chapters. **Chapter 1** establishes the theoretical foundations necessary for understanding the AI techniques employed throughout this work, covering the core concepts of machine learning, deep learning, convolutional neural networks, transfer learning architectures such as EfficientNet-B3 and ResNet-50, fuzzy logic, and the broader role of artificial intelligence in modern agriculture.

**Chapter 2** is dedicated to the design and experimental evaluation of the proposed AI solutions. It presents three deep learning models — a custom CNN, ResNet-50, and EfficientNet-B3 — trained and benchmarked on a multi-class plant disease classification dataset spanning lettuce, strawberry, and tomato crops. In addition, a YOLO11-based instance segmentation model is developed for tomato detection, providing fine-grained spatial analysis of plant health. The chapter concludes with a comparative discussion of all models in terms of accuracy, generalisation, and computational cost.

**Chapter 3** introduces the digital twin of the proposed system. By constructing a virtual replica of the modular, connected, and hydroponic infrastructure, this chapter bridges the gap between physical deployment and simulation-based monitoring. The digital twin enables real-time state estimation, predictive maintenance, and scenario testing without interrupting the physical system, constituting a critical step toward a fully autonomous and intelligent agricultural platform. Finally, a general conclusion synthesises the contributions of this work, reflects on the results obtained, and outlines promising directions for future research and development.

## FUNDAMENTALS OF ARTIFICIAL INTELLIGENCE FOR SMART AGRICULTURE

### Introduction

Artificial Intelligence has become an important tool in modern agriculture, especially for improving crop monitoring, disease detection, yield estimation, and automatic decision-making. In hydroponic systems, where plant growth depends on the precise control of parameters such as pH, nutrient concentration, temperature, and humidity, intelligent techniques can significantly improve system efficiency and crop quality.

This chapter presents the theoretical foundations required for the proposed intelligent hydroponic system. It introduces the main concepts of Artificial Intelligence, Machine Learning, Deep Learning, Convolutional Neural Networks, transfer learning, object detection, YOLO models, evaluation metrics, and fuzzy logic. These concepts provide the necessary background for understanding the AI models, fuzzy controller, digital twin, and HydroBot assistant developed in the following chapters.

## **Part 1: Artificial Intelligence**

Artificial Intelligence (AI) refers to the ability of machines and computer systems to perform tasks that normally require human intelligence, such as learning, reasoning, and decision-making. AI systems can analyze large amounts of data, recognize patterns, and improve their performance over time.

The concept of AI was introduced in 1956 during the Dartmouth Conference. Since then, it has evolved significantly due to advances in computing power, data availability, and algorithm development.

Today, AI includes several fields such as machine learning, deep learning, computer vision, and natural language processing, and it is widely used in many sectors including agriculture.

## **1 Machine Learning**

Machine Learning (ML) is a subfield of Artificial Intelligence that allows computers to learn from data without being explicitly programmed. Instead of following fixed rules, ML algorithms identify patterns in data and use them to make predictions or decisions.

Machine learning is widely used in many fields such as healthcare, finance, and agriculture for tasks like prediction, classification, and pattern recognition.

Machine learning methods are generally divided into three main categories: supervised learning, unsupervised learning, and reinforcement learning.

### **1.1 Supervised Learning**

Supervised learning is a machine learning approach where the model is trained using labeled data. This means that each input data has a corresponding correct output.

The goal of the algorithm is to learn the relationship between input and output in order to make accurate predictions on new data.

Common applications include image classification, crop disease detection, and yield prediction.

### **1.2 Unsupervised Learning**

Unsupervised learning is used when the data does not have labeled outputs. The algorithm tries to identify hidden patterns or structures in the data.

It is often used for clustering, grouping similar data points together.

In agriculture, it can be used to group similar soil conditions or crop patterns.

### **1.3 Reinforcement Learning**

Reinforcement learning is a type of machine learning where an agent learns by interacting with an environment. The agent receives rewards or penalties based on its actions.

Over time, the agent learns to choose the actions that maximize the total reward.

This approach is often used in robotics, automation, and decision-making systems.

## **2 Deep Learning**

Deep learning is a specialized branch of machine learning based on artificial neural networks with multiple layers. These networks can automatically learn complex features from large datasets.

Deep learning has achieved significant success in fields such as image recognition, speech processing, and autonomous systems.

In agriculture, deep learning is often used for plant disease detection, crop monitoring, and yield estimation.

## 2.1 Neural network vocabulary

In order to understand the training process of deep learning models, particularly Convolutional Neural Networks (CNNs), it is necessary to define several key concepts commonly used during model development. These concepts describe how data are processed, how the model learns from examples, and how its parameters are updated in order to improve prediction performance.

### Cost Function

When we build a network, the network tries to predict the output as close as possible to the actual value. We measure this accuracy of the network using the **cost/loss function**. The main idea is to calculate a **distance between prediction and target**, as illustrated in Figure 1.1.

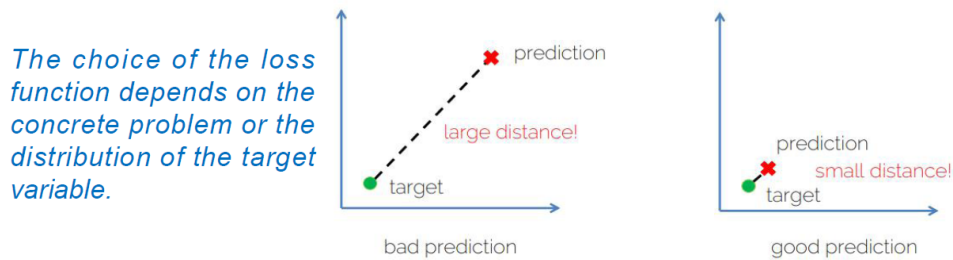


Figure 1.1: Cost function principle based on the distance between prediction and target

### Gradient descent

Gradient descent is an optimization algorithm used to minimize some function by iteratively moving in the direction of steepest descent as defined by the negative of the gradient. In machine learning, we use gradient descent **to update the parameters of our model**. Parameters refer to **weights** in neural networks. To put it simply, we use gradient descent to minimize the cost function,  $J(\mathbf{w})$  as illustrated in Figure 1.2.

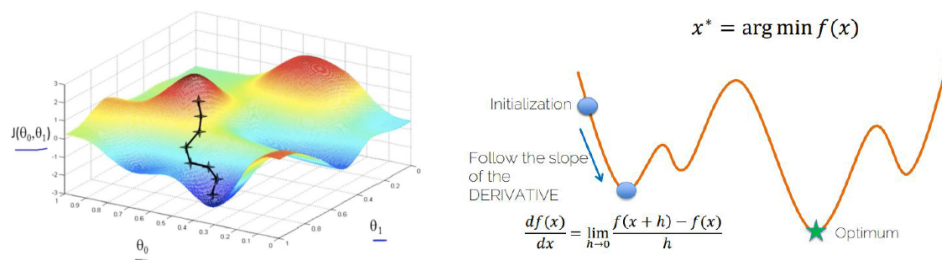


Figure 1.2: Gradient descent principle for minimizing the cost function

## Learning rate

Learning Rate: is defined as **the amount of minimization in the cost function in each iteration**. In simple terms, the rate at which we descend towards the minima of the cost function is the learning rate. **We should choose the learning rate very carefully** since it should neither be very large that the optimal solution is missed and nor should be very low that it takes forever for the network to converge as illustrated in Figure 1.3.

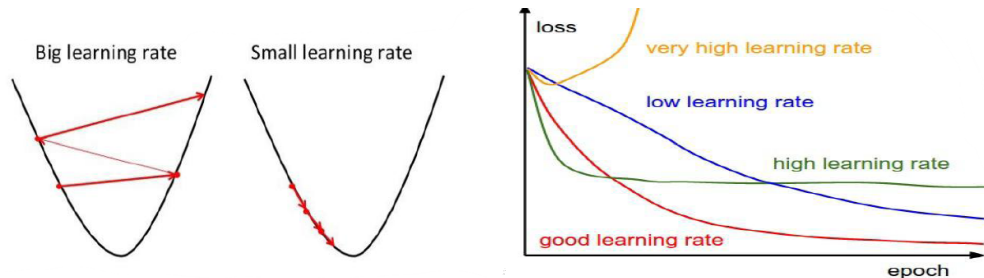


Figure 1.3: Learning rate

## Backpropagation

Backpropagation after we calculated the error of the network, this error is then fed back to the network along with the gradient of the cost function to update the weights of the network. These weights are then updated so that the errors in the subsequent iterations is reduced. This updating of weights using the gradient of the cost function is known as backpropagation, as shown in Figure 1.4.

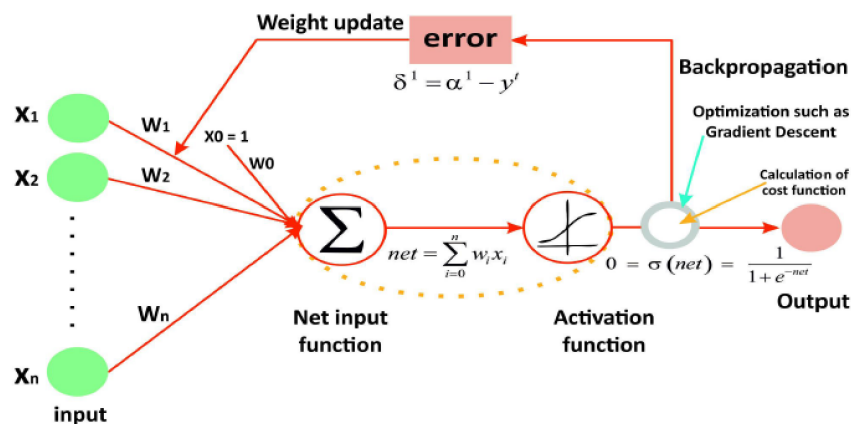


Figure 1.4: Backpropagation

## Batches

Batches while training a neural network, instead of sending the entire input in one go, **we divide in input into several chunks of equal size randomly**. Training the data on batches makes the model **more generalized** as compared to the model built when the entire data set is fed to the network in one go.

- *It requires less memory.*
- *Typically networks train faster with mini-batches.*
- **Batch Gradient Descent** : Batch Size = Size of Training Set

- **Stochastic Gradient Descent:** Batch Size = 1
- **Mini-Batch Gradient Descent:**  $1 < \text{Batch Size} < \text{Size of Training Set}$

## Epochs

Epoch is defined as a **single training iteration** of all batches in both **forward and backpropagation**. This means 1 epoch is a single forward and backward pass of the entire input data. The number of epochs you would use to train your network can be chosen **by you**. It is highly likely that *more number of epochs would show higher accuracy* of the network, however, it would also take longer for the network to converge. Also, you must take care that if the number of epochs are too high, the network might be **over-fit**. Example :

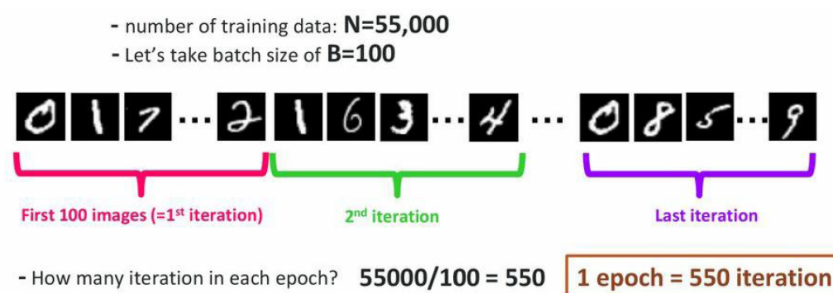


Figure 1.5: Example of epochs

## Updating weights

Updating weights In a neural network, weights are updated as follows:

- Step 1: Take a batch of training data.
- Step 2: Perform forward propagation to obtain the corresponding loss.
- Step 3: Backpropagate the loss to get the gradients.
- Step 4: Use the gradients to update the weights of the network.

The concepts presented above are fundamental for understanding how deep learning models are trained and optimized. However, when the input data are images, classical neural networks are not always sufficient, because they do not explicitly take into account the spatial structure of images. For this reason, Convolutional Neural Networks (CNNs) are widely used in computer vision tasks. CNNs are designed to automatically extract relevant visual features such as edges, textures, shapes, and more complex patterns through successive layers. This makes them particularly suitable for image classification and object detection applications, including agricultural systems where images are used to identify plant diseases, monitor crop growth, or detect visual anomalies.

## 3 CNN architecture

After introducing the main concepts related to neural networks and their training process, it is important to focus on a specific deep learning architecture widely used in image analysis: Convolutional Neural Networks, commonly known as CNNs. CNNs have played a major role in the development of deep learning, especially for visual recognition tasks [4,5].

CNNs are particularly effective for image classification and object detection tasks because they are able to automatically extract relevant visual features from images. Unlike traditional machine learning approaches, which often require manual feature extraction, CNNs learn these features directly from the data through several layers [4]. These layers generally include convolutional layers, activation functions, pooling layers, and fully connected layers . Figure 1.6 illustrates the general architecture of a Convolutional Neural Network.

In agricultural applications, CNNs have shown strong performance in tasks such as plant disease detection, fruit classification, weed identification, and crop monitoring [6,7]. Their ability to analyze images makes them highly suitable for detecting visual symptoms on plant leaves, such as color changes, spots, lesions, and texture variations.

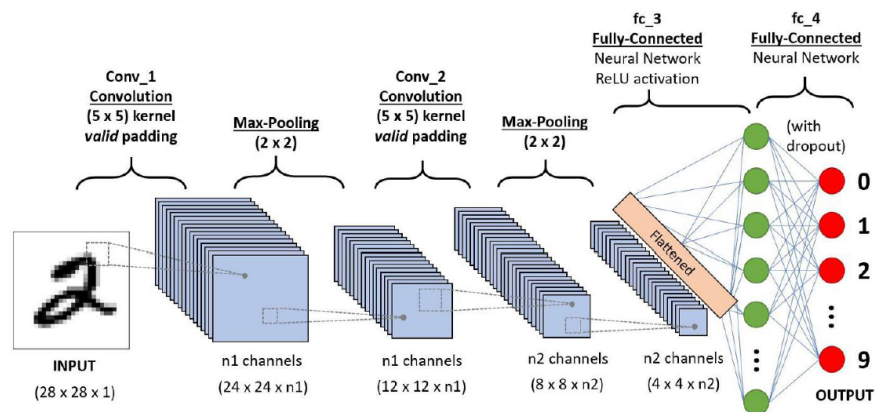


Figure 1.6: CNN architecture

## Fully Connected Layers

Fully connected layers are usually located at the end of CNN architectures. Before reaching these layers, the multidimensional feature maps generated by convolutional and pooling layers are transformed into a one-dimensional vector through a **flattening** operation. This vector contains the most relevant features extracted from the input image.

As shown in Figure 1.7, each neuron in a fully connected layer is connected to all neurons from the previous layer. These layers combine the extracted features and use them to perform the final decision. In classification tasks, the last fully connected layer produces the output probabilities corresponding to the different classes.

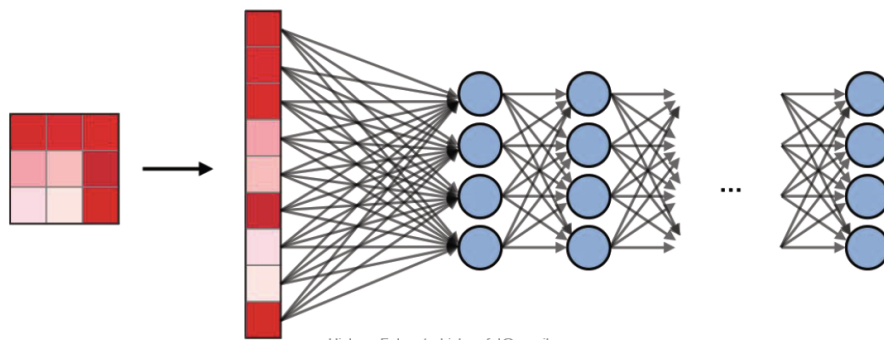


Figure 1.7: Fully Connected Layer

The choice of the output layer depends on the type of classification problem. As shown in Table 1.1, the number of output neurons, the activation function, and the loss function vary

according to whether the problem is binary classification, multilabel binary classification, or multiclass classification.

Table 1.1: Hyperparameters according to the classification problem

| Hyperparameter                         | Binary classification  | Multilabel binary classification | Multiclass classification |
|----------------------------------------|------------------------|----------------------------------|---------------------------|
| Input neurons                          | One per input feature  | One per input feature            | One per input feature     |
| Hidden layers neurons per hidden layer | Depends on the problem | Depends on the problem           | Depends on the problem    |
| Output neurons                         | 1                      | 1 per label                      | 1 per class               |
| Hidden activation                      | ReLU (relu)            | ReLU (relu)                      | ReLU (relu)               |
| Output layer activation                | Logistic (sigmoid)     | Logistic (sigmoid)               | Softmax (softmax)         |
| Loss function                          | binary_crossentropy    | binary_crossentropy              | Cross entropy             |

## Mathematical Formulation

### Convolution

$$Y = X * W + b \quad (1.1)$$

where  $X$  represents the input image or feature map,  $W$  denotes the convolution kernel,  $b$  is the bias term, and  $Y$  is the resulting feature map.

### ReLU Activation

After the convolution operation, a non-linear activation function is applied. The most commonly used activation function in CNNs is the Rectified Linear Unit (ReLU), defined as:

$$f(x) = \max(0, x) \quad (1.2)$$

ReLU introduces non-linearity and mitigates vanishing gradient problems.

### Logistic (Sigmoid) Activation

The logistic function, also known as the sigmoid function, is an activation function commonly used in binary classification problems. It transforms any real-valued input into a value between 0 and 1, which can be interpreted as a probability. It is defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad (1.3)$$

where  $x$  represents the input value, and  $\sigma(x)$  represents the output of the sigmoid function.

The sigmoid function is mainly used in the output layer of a neural network when the classification task involves two classes. However, for multi-class classification problems, the Softmax function is generally preferred because it produces a probability distribution over all classes.

## Softmax Function

For multi-class classification, the Softmax function is used in the output layer. It converts the output logits into probabilities for each class. In this work, since the classification problem involves twenty classes, the Softmax function is expressed as:

$$P(y_i) = \frac{e^{z_i}}{\sum_{j=1}^{20} e^{z_j}} \quad (1.4)$$

Softmax converts the raw logits into a probability distribution across all twenty classes.

where  $z_i$  is the logit corresponding to class  $i$ , and  $P(y_i)$  represents the predicted probability of class  $i$ .

## Categorical Cross-Entropy Loss

This loss measures divergence between predicted and true distributions.

$$L = - \sum_{i=1}^{20} y_i \log(\hat{y}_i) \quad (1.5)$$

where  $y_i$  is the true label of class  $i$ , and  $\hat{y}_i$  is the predicted probability for that class. The objective of the training process is to minimize this loss in order to improve the classification performance of the model.

## Convolutional Layers

The convolutional layer is the core component of a CNN. Its main role is to extract features from the input image using small filters called kernels. A kernel slides over the input image and performs element-wise multiplications followed by a summation. The result of this operation forms a feature map that highlights specific visual patterns.

Figure 1.8 illustrates an example of a convolution operation. The kernel is applied to a small region of the image, and the resulting value is placed in the corresponding position of the output feature map. By using several kernels, the convolutional layer can detect different types of features, such as edges, contours, textures, or local structures.

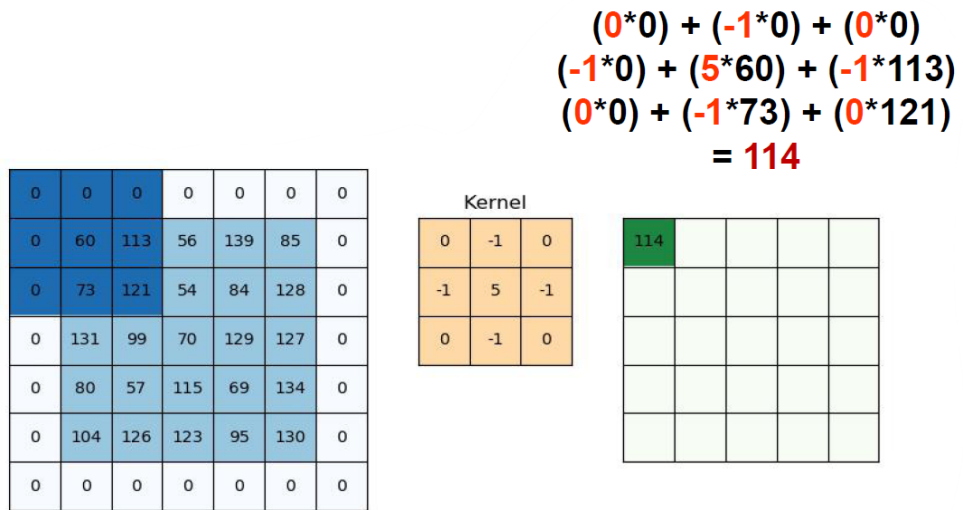


Figure 1.8: Convolution process in a CNN using a 3×3 kernel to extract image features

Several parameters control the behavior of a convolutional layer. The kernel size defines the dimensions of the filter. The stride defines the step size used when the kernel moves over the image. Padding is used to control the spatial size of the output feature map. When valid padding is used, no additional pixels are added around the image, which reduces the output size. When same padding is used, zeros are added around the image in order to preserve the spatial dimensions after convolution.

The output size of a convolutional layer can be calculated using the following formula:

$$o = \left\lceil \frac{W - K + 2P}{S} \right\rceil + 1 \quad (1.6)$$

where:

- (O) is the output size,
- (W) is the input size,
- (K) is the kernel size,
- (P) is the padding
- (S) is the stride.

This formula is useful for determining the dimensions of feature maps after convolution.

## Pooling Layers

Pooling layers are generally used after convolutional layers. Their main objective is to reduce the spatial dimensions of feature maps while preserving the most important information. This operation reduces the computational cost, decreases memory usage, and limits the risk of overfitting. As shown in Figure 1.9, max pooling selects the maximum value from each local region of the feature map. For example, using a  $(2 \times 2)$  pooling window with a stride of 2 reduces the width and height of the feature map by half. This operation keeps the strongest activation in each region, which often corresponds to the most important detected feature.

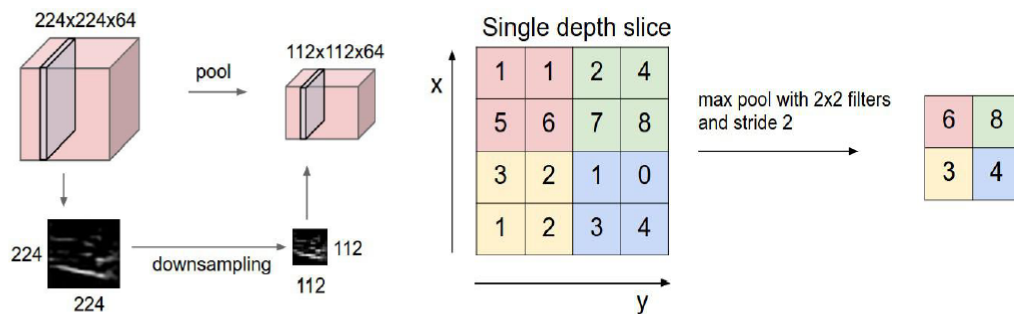


Figure 1.9: Principle of max pooling in a CNN.

Pooling also introduces a certain level of translation invariance. This means that small shifts or changes in the position of an object in the image do not significantly affect the final prediction. This property is very important in real-world applications because objects or symptoms may appear in different positions within images.

There are different types of pooling operations. The most commonly used are max pooling and average pooling. Max pooling keeps the maximum value from each region,

while average pooling calculates the average value of the selected region. Figure 1.10 compares these two operations.

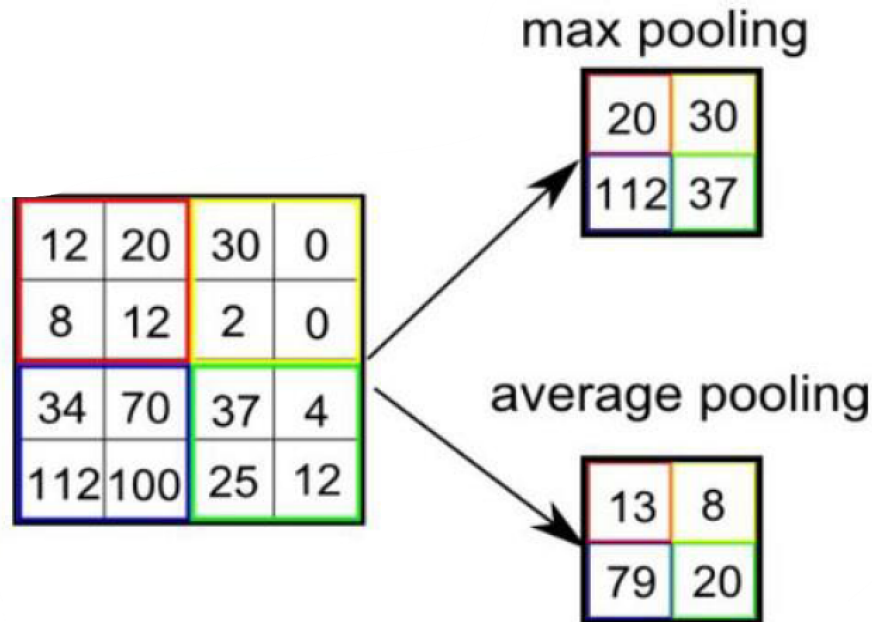


Figure 1.10: Comparison between max pooling and average pooling

Max pooling is often preferred in CNN architectures because it preserves the strongest and most discriminative features. Average pooling, on the other hand, provides a smoother representation by considering all values in the selected region. The choice between these two methods depends on the nature of the task and the architecture of the model.

### 3.1 Transfer Learning

Training deep learning models, especially Convolutional Neural Networks (CNNs), usually requires a very large amount of labeled data and high computational resources. In many real-world applications, collecting and annotating large datasets is difficult, time-consuming, and expensive. Moreover, training a CNN from scratch may lead to poor performance when the available dataset is small, because the model may not learn enough representative features and can easily overfit the training data.

For this reason, transfer learning appeared as an effective solution to reduce the need for large datasets and long training times. The main idea is to reuse the knowledge learned by a model trained on a large dataset and adapt it to a new but related problem. Instead of starting the learning process from zero, the model starts from previously learned features such as edges, corners, textures, shapes, and object parts.

Transfer learning is strongly related to CNNs because CNN architectures are widely used for image processing tasks. In a CNN, the first convolutional layers generally learn low-level visual features such as edges and simple textures. The intermediate layers learn more complex patterns, while the deeper layers learn task-specific features. Since the low-level and intermediate features are often useful for many image-related problems, they can be transferred from one task to another.

For example, a CNN trained on a large dataset such as ImageNet<sup>1</sup> can learn general visual representations from millions of images. These learned representations can then be reused for a new classification task with fewer images. The final layers of the CNN are usually replaced or adapted according to the number of classes in the new problem. The model can then be used either as a feature extractor or fine-tuned by retraining some layers on the new dataset.

Therefore, transfer learning allows CNN-based models to achieve better performance with less data, less training time, and lower computational cost. It is particularly useful in practical applications where building a large labeled dataset is difficult.

## 4 ResNet

ResNet is a powerful CNN architecture commonly used in transfer learning for image classification tasks. Its main idea is based on skip connections, also called shortcut connections, which allow information to pass directly from one layer to another. This helps reduce training difficulties in very deep networks and improves gradient flow. As shown in Figure 1.11, ResNet is composed of convolutional blocks and identity blocks that make the network easier to train and more efficient.

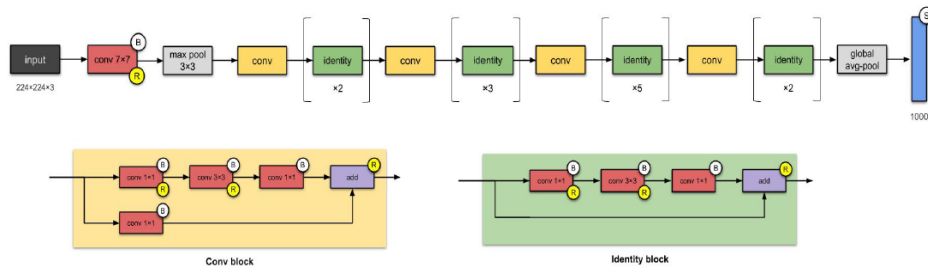


Figure 1.11: ResNet architecture based on convolutional and identity blocks with skip connections

### 4.1 EfficientNet-B3

After presenting ResNet as a deep CNN architecture based on residual connections, another important architecture used in transfer learning is EfficientNet-B3. EfficientNet-B3 belongs to the EfficientNet family, which includes several models from B0 to B7. These models were designed to improve the balance between accuracy and computational cost. The main idea of EfficientNet is based on compound scaling. Unlike traditional CNN architectures that increase only one dimension of the network, such as depth, width, or input image resolution, EfficientNet scales these three dimensions together in a balanced way. Depth refers to the number of layers, width refers to the number of channels or feature maps, and resolution refers to the size of the input image.

EfficientNet-B3 is considered more efficient than many classical CNN models because it can achieve good performance while using fewer computational resources. This makes it suitable for applications where memory, energy consumption, and training time are important constraints. For this reason, EfficientNet-B3 is often used as a pre-trained model in transfer learning.

<sup>1</sup>ImageNet is a large-scale image dataset widely used in computer vision. It contains millions of labeled images organized into thousands of categories and is commonly used to train and evaluate deep learning models.

As shown in Figure 1.12, EfficientNet-B3 starts with a stem convolution layer, followed by several MBConv blocks. These blocks are based on inverted residual and linear bottleneck layers, depth-wise separable convolutions, and squeeze-and-excitation mechanisms. The final part of the model contains a convolution layer, pooling, and a classifier that produces the final output.

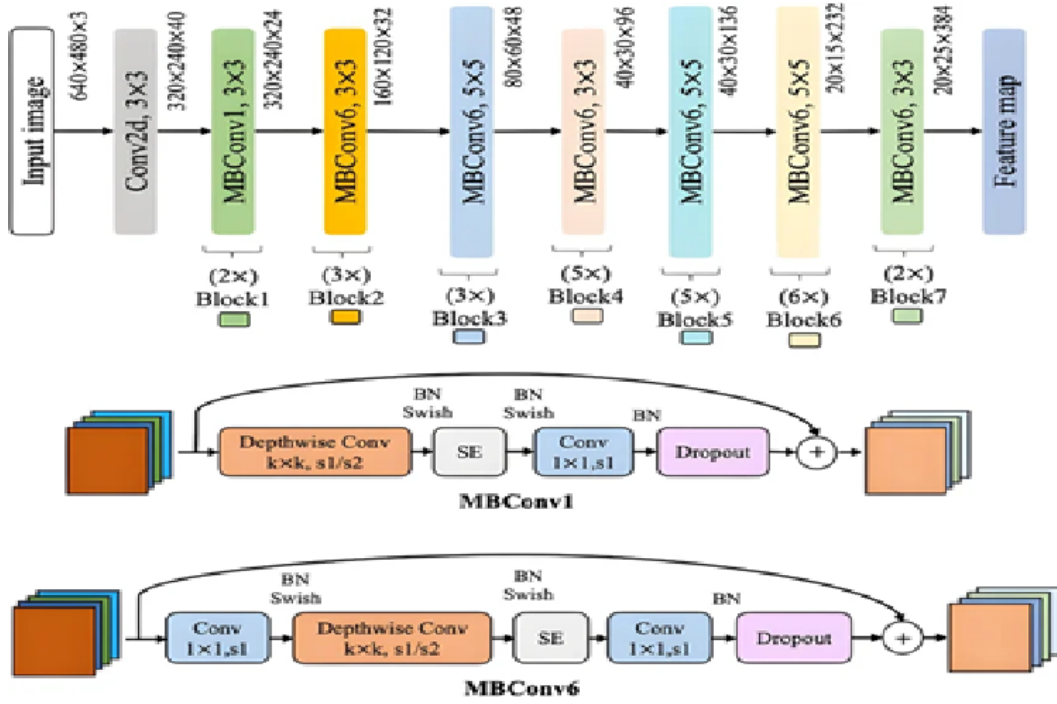


Figure 1.12: EfficientNet architecture based on MBConv blocks

**NB.** MBConv blocks, also called Mobile Inverted Bottleneck Convolution blocks, are the main building units of EfficientNet architectures. They are based on inverted residual connections and depthwise separable convolutions. Their objective is to extract relevant image features while reducing the computational cost. In EfficientNet-B3, these blocks allow the model to achieve good accuracy with fewer parameters and lower memory consumption compared to classical CNN architectures.

## 5 Object detection and Segmentation

After presenting transfer learning and its importance in CNN-based image classification, it is necessary to extend the discussion to more advanced computer vision tasks. In image classification, the model assigns one label to the entire image. However, in many real-world applications, knowing only the class of an image is not sufficient. It is also important to know where the object is located and, in some cases, to identify its exact shape within the image.

This leads to two important tasks in computer vision: object detection and image segmentation. Object detection aims to identify and localize objects in an image by predicting bounding boxes around them. Each detected object is associated with a class label and a confidence score. In contrast, image segmentation provides a more detailed analysis by assigning a label to each pixel of the image.

As shown in Figure 1.13, image classification, object detection, and segmentation differ in the level of information they provide. Classification predicts the category of the image, object detection predicts both the category and the location of objects, while segmentation identifies the regions or pixels belonging to each object.

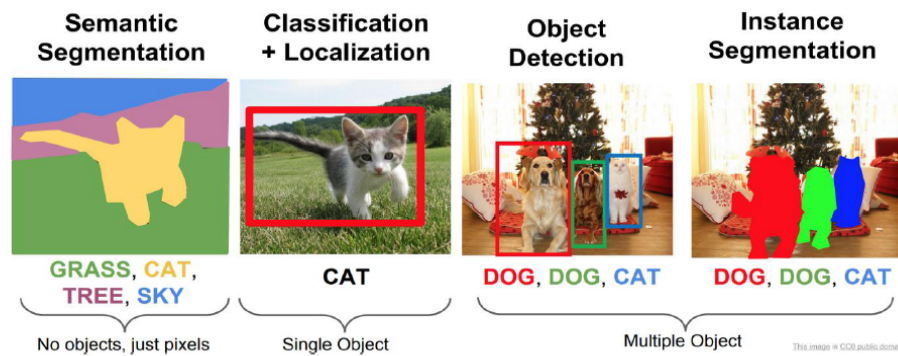


Figure 1.13: Comparison between image classification, object detection, and image segmentation

Several deep learning models have been proposed for object detection and segmentation tasks. Among these models, YOLO is one of the most widely used approaches because of its speed and ability to perform real-time object detection.

## 5.1 What is Yolo?

**You Only Look Once (YOLO)** is a deep learning algorithm based on **convolutional neural networks (CNNs)** and used for **object detection** [8]. Unlike traditional detection methods that analyze an image in several steps, YOLO processes the entire image only once. This allows the model to detect objects, predict their **bounding boxes**, and assign their **class labels** in a single forward pass.

YOLO is considered one of the fastest object detection algorithms. It is especially useful when **real-time detection** is required, because it provides a good balance between **speed** and **accuracy** [8]. Several versions of YOLO have been proposed over time, such as **YOLOv3** [9] and **...YOLO4** [10]. Each version improves the detection performance, speed, or accuracy of the model.

## 5.2 Evolution of YOLO Models

Table 1.2 illustrates the progression of YOLO models from their inception to the most recent versions. Each iteration has brought significant improvements in object detection capabilities, computational efficiency, and versatility in handling various computer vision tasks.

Table 1.2: YOLO: Evolution of models

| Release     | Year | Tasks                                                                               | Contributions                                     | Framework |
|-------------|------|-------------------------------------------------------------------------------------|---------------------------------------------------|-----------|
| YOLO [8]    | 2015 | Object detection, basic classification                                              | Single-stage object detector                      | Darknet   |
| YOLOv2      | 2016 | Object detection, improved classification                                           | Multi-scale training, dimension clustering        | Darknet   |
| YOLOv3 [9]  | 2018 | Object detection, multi-scale detection                                             | SPP block, Darknet-53 backbone                    | Darknet   |
| YOLOv4 [10] | 2020 | Object detection, basic object tracking                                             | Mish activation, CSPDarknet-53 backbone           | Darknet   |
| YOLOv5      | 2020 | Object detection, basic instance segmentation                                       | Anchor-free detection, SWISH activation, PANet    | PyTorch   |
| YOLOv6      | 2022 | Object detection, instance segmentation                                             | Self-attention, anchor-free object detection      | PyTorch   |
| YOLOv7      | 2022 | Object detection, object tracking, instance segmentation                            | Transformers, E-ELAN reparameterisation           | PyTorch   |
| YOLOv8      | 2023 | Object detection, instance segmentation, panoptic segmentation, keypoint estimation | GANs, anchor-free detection                       | PyTorch   |
| YOLOv9      | 2024 | Object detection, instance segmentation                                             | PGI and GELAN                                     | PyTorch   |
| YOLOv10     | 2024 | Object detection                                                                    | Consistent dual assignments for NMS-free training | PyTorch   |

### 5.3 YOLO 11 Architecture

The evolution of the YOLO algorithm continued with the introduction of **YOLOv11**, which represents a recent advancement in **real-time object detection**. This version builds on the strengths of previous YOLO models while improving **accuracy**, **efficiency**, and **adaptability**. Unlike earlier versions mainly focused on object detection, YOLOv11 supports a wider range of computer vision tasks, including **object detection**, **instance segmentation**, and **pose estimation**. This makes it suitable for several real-world applications where fast and accurate visual analysis is required. Therefore, YOLOv11 reflects the continuous progress of YOLO models toward more powerful, flexible, and practical solutions for computer vision applications [1].

The **YOLO framework** revolutionized **object detection** by introducing a unified neural network architecture that simultaneously performs **bounding box regression** and **object classification** tasks [11]. This integrated approach represents an important difference from traditional two-stage detection methods, since YOLO allows **end-to-end training** through a fully differentiable architecture. At its core, the YOLO architecture is composed of three

main components: the **backbone**, the **neck**, and the **head**. First, the backbone acts as the main **feature extractor**. It uses convolutional neural networks to transform the input image into feature maps at different scales. Second, the neck serves as an intermediate processing stage that combines and improves the extracted features in order to better detect objects of different sizes. Finally, the head is responsible for the final prediction. It generates the **bounding boxes**, **class labels**, and confidence scores required for object localization and classification.

Building on this general architecture, **YOLO11** extends the foundation of previous YOLO versions, especially YOLOv8, by introducing architectural improvements and parameter optimizations. These modifications aim to improve detection performance while maintaining the real-time capability of the model, as illustrated in Figure 1.14.

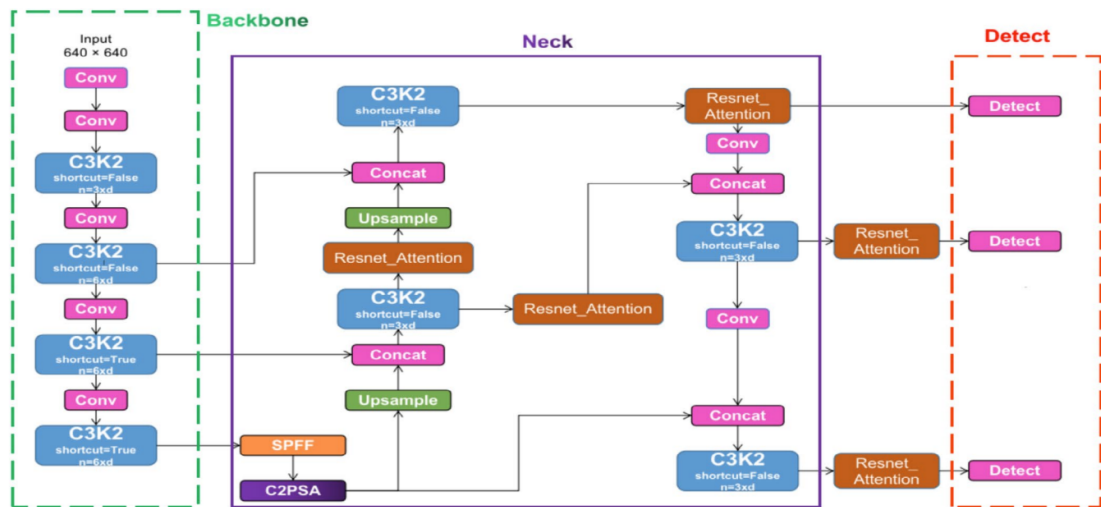


Figure 1.14: Architecture of YOLOv11 [1]

- **Backbone**

The **backbone** is responsible for extracting important visual features from the input image. In YOLO11, the input image first passes through several convolutional layers. These layers reduce the spatial size of the image while increasing the number of channels. This allows the model to learn useful information such as edges, textures, shapes, and object parts.

A major improvement in YOLO11 is the use of the **C3k2 block**, which replaces the **C2f block** used in previous versions such as YOLOv8. The C3k2 block is more computationally efficient because it uses smaller convolution kernels, which helps improve processing speed while maintaining good detection performance [12].

- **SPPF and C2PSA Blocks**

YOLO11 keeps the **SPPF** block, which means **Spatial Pyramid Pooling Fast**. This block helps the model extract features at different spatial scales. In addition, YOLO11 introduces the **C2PSA** block, which stands for **Cross Stage Partial with Spatial Attention**. This block allows the model to focus more on the most important regions of the image. As a result, the model can better detect objects with different sizes and positions [12].

- **Neck**

The **neck** is the intermediate part between the backbone and the head. Its role is to combine feature maps extracted at different levels of the network. This is important

because some objects may appear large, while others may be small or partially hidden. By combining multi-scale features, the neck improves the ability of YOLO11 to detect objects more accurately.

In YOLO11, the neck also uses the **C3k2 block** after operations such as upsampling and concatenation. This improves the efficiency of feature fusion and helps the model maintain a good balance between speed and accuracy [12].

- **Attention Mechanism**

Another important improvement in YOLO11 is the use of a **spatial attention mechanism** through the **C2PSA module**. This mechanism helps the model give more importance to useful regions in the image and ignore less relevant information. This can improve detection performance, especially for small objects or objects that are difficult to distinguish from the background [12].

- **Head**

The **head** is the final part of the YOLO11 architecture. It receives the refined feature maps from the neck and produces the final detection results. These results include the **bounding box coordinates**, the **objectness score**, and the **class scores**. Therefore, the head is responsible for both object localization and classification.

In YOLO11, several **C3k2 blocks** are used in the head to further process and refine the feature maps. These blocks help the model extract more detailed information before making the final predictions [12].

- **CBS Blocks**

The head also contains **CBS blocks**, which refer to **Convolution-Batch Normalization-SiLU** layers. These blocks are used to refine the feature maps before prediction. The convolution operation extracts relevant features, batch normalization stabilizes the learning process, and the **SiLU activation function** introduces non-linearity to improve the model's performance [13].

- **Final Detection Layer**

The final detection layer produces the output of the YOLO11 model. It predicts the **bounding boxes** used to locate objects in the image, the **objectness scores** indicating the presence of objects, and the **class probabilities** used to identify the category of each detected object. This allows YOLO11 to perform object detection in a fast and efficient way.

## 6 Evaluation Metrics

Evaluation metrics are used to measure the performance of a machine learning model after the training phase. They allow us to determine whether the model is able to make correct predictions on new and unseen data. The choice of the evaluation metric depends mainly on the type of problem, such as classification, regression, or object detection.

In classification problems, the most commonly used metrics are accuracy, precision, recall, and F1-score. Accuracy measures the proportion of correctly classified samples among all samples. However, accuracy alone may not be sufficient, especially when the dataset is imbalanced. For this reason, precision and recall are often used to provide a more detailed evaluation of the model performance.

Precision measures the proportion of correctly predicted positive samples among all samples predicted as positive. Recall measures the proportion of correctly detected positive

samples among all real positive samples. The F1-score combines precision and recall into a single metric by calculating their harmonic mean. This metric is particularly useful when there is a need to balance false positives and false negatives.

For regression problems, evaluation metrics are different because the objective is to predict continuous numerical values. Common regression metrics include mean absolute error, mean squared error, and the coefficient of determination, also known as the  $R^2$  score. These metrics measure the difference between the predicted values and the actual values.

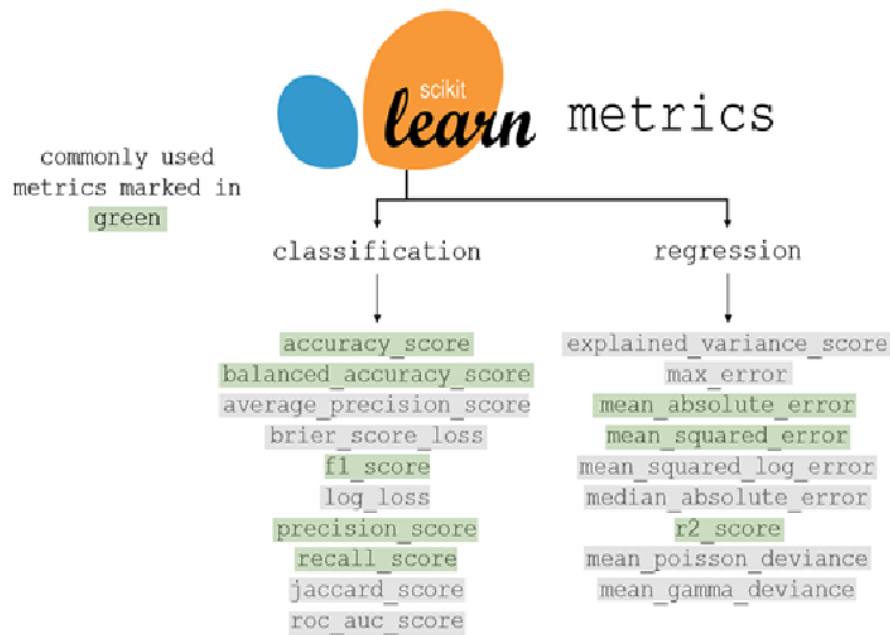


Figure 1.15: Common evaluation metrics used for classification and regression tasks [2]

As illustrated in Figure 1.15, Scikit-learn provides several evaluation metrics for both classification and regression tasks. In this work, classification metrics such as accuracy, precision, recall, and F1-score are mainly considered to evaluate the performance of the proposed model.

## 6.1 Classification Metrics

Since our problem is a classification problem, the evaluation of the model is based on classification metrics. These metrics are computed from the confusion matrix, which compares the predicted classes with the actual classes. As shown in Figure 1.16, the confusion matrix is composed of four main elements: true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN).

|              |          | Predicted Class                            |                                                            |                                                          |
|--------------|----------|--------------------------------------------|------------------------------------------------------------|----------------------------------------------------------|
|              |          | Positive                                   | Negative                                                   |                                                          |
| Actual Class | Positive | True Positive (TP)                         | False Negative (FN)<br><b>Type II Error</b>                | <b>Sensitivity</b><br>$\frac{TP}{(TP + FN)}$             |
|              | Negative | False Positive (FP)<br><b>Type I Error</b> | True Negative (TN)                                         | <b>Specificity</b><br>$\frac{TN}{(TN + FP)}$             |
|              |          | <b>Precision</b><br>$\frac{TP}{(TP + FP)}$ | <b>Negative Predictive Value</b><br>$\frac{TN}{(TN + FN)}$ | <b>Accuracy</b><br>$\frac{TP + TN}{(TP + TN + FP + FN)}$ |

Figure 1.16: Confusion matrix for classification performance evaluation [3]

Based on these values, several performance metrics can be calculated, such as accuracy, precision, recall, specificity, and F1-score. Accuracy measures the overall performance of the model, while precision evaluates the correctness of positive predictions. Recall, also called sensitivity, measures the ability of the model to detect positive samples. Specificity evaluates the ability of the model to correctly identify negative samples. The F1-score combines precision and recall and is particularly useful when the dataset contains imbalanced classes.

Table 1.3: Main classification metrics derived from the confusion matrix

| Metric             | Formula                             | Interpretation                                      |
|--------------------|-------------------------------------|-----------------------------------------------------|
| Accuracy           | $\frac{TP + TN}{TP + TN + FP + FN}$ | Overall performance of the model.                   |
| Precision          | $\frac{TP}{TP + FP}$                | Measures how accurate the positive predictions are. |
| Recall/Sensitivity | $\frac{TP}{TP + FN}$                | Measures the coverage of actual positive samples.   |
| Specificity        | $\frac{TN}{TN + FP}$                | Measures the coverage of actual negative samples.   |
| F1-score           | $\frac{2TP}{2TP + FP + FN}$         | Hybrid metric useful for imbalanced classes.        |

## 6.2 Object Detection Metrics

In object detection tasks, evaluating the model is different from simple classification. The model must not only predict the correct class of an object, but also localize it correctly using a bounding box. Therefore, metrics such as the mean squared error (MSE) may be useful during training, but they are not sufficient to evaluate how well the model predicts bounding boxes.

One of the most important metrics used for object detection is the Intersection over Union (IoU). IoU measures the overlap between the predicted bounding box and the ground-truth

bounding box. A high IoU value means that the predicted box is very close to the real object location, while a low IoU value indicates poor localization.

$$IoU = \frac{B_{pred} \cap B_{gt}}{B_{pred} \cup B_{gt}} \quad (1.7)$$

where  $B_{pred}$  represents the predicted bounding box and  $B_{gt}$  represents the ground-truth bounding box.

Another important metric for object detection is the mean Average Precision (mAP). In order to calculate mAP, precision-recall curves are generated at different IoU thresholds. For example, COCO<sup>2</sup> is commonly used to evaluate object detection models such as YOLO., the IoU threshold ranges from 0.5 to 0.95 with a step of 0.05, which is commonly represented as AP@[.5:.05:.95]. The mAP metric gives a global evaluation of the detection performance by considering both classification accuracy and localization quality.

$$Recall = \frac{TP}{TP + FN} \quad (1.8)$$

$$Precision = \frac{TP}{TP + FP} \quad (1.9)$$

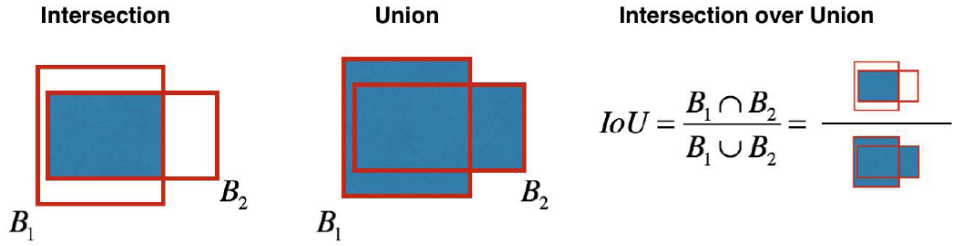


Figure 1.17: Evaluation metrics for object detection using IoU and mAP

As shown in Figure 1.17, object detection models are mainly evaluated using IoU and mAP. These metrics are particularly important for YOLO-based models, because they evaluate both the ability of the model to detect the correct object class and the quality of the predicted bounding box.

## 7 State of the Art of Artificial Intelligence in Agriculture and Hydroponics

The increasing demand for food production has become one of the major global challenges of the twenty-first century. This situation is mainly driven by population growth, climate change, reduction of arable land, water scarcity, and pressure on natural resources. According to Mir et al. [14], vertical farming is increasingly considered a promising agricultural approach to overcome the limitations of conventional farming, especially in regions affected by land scarcity and environmental constraints. In parallel, water availability has become a critical issue for agriculture. Bhanja et al. [15] highlighted the importance of monitoring groundwater storage, particularly in large river basins where climate variability and intensive water use can significantly affect water availability. These challenges emphasize

<sup>2</sup>COCO, which stands for Common Objects in Context, is a large-scale dataset widely used in computer vision for object detection, segmentation, and image captioning tasks.

the need for alternative cultivation systems capable of producing crops efficiently while reducing land and water consumption.

Hydroponic vertical farming has emerged as an effective solution for sustainable crop production. Unlike traditional soil-based agriculture, hydroponic systems use nutrient-rich water solutions to supply plants with the required minerals. This technique allows crops to be grown in controlled environments, independently of seasonal and climatic variations. Kobayashi et al. [16] discussed vertical farming as a system that can reduce agricultural land requirements, although it may require additional energy resources. Similarly, Ruffi-Salis et al. [17] showed that hydroponic systems can contribute to more efficient nutrient management and resource recovery in urban agriculture. These characteristics make hydroponics particularly suitable for intensive production in limited spaces, such as greenhouses, urban farms, and indoor vertical farms.

Several crops can be cultivated in hydroponic and controlled-environment systems, including lettuce, tomato, strawberry, basil, and rice. Lettuce is one of the most widely used crops in hydroponic farming because of its short growth cycle, high economic value, and sensitivity to controlled environmental conditions. Eshkabilov et al. [18] demonstrated the potential of hyperspectral imaging for evaluating nutrient content of hydroponically grown lettuce cultivars. Tomato is also an important crop in smart agriculture because its leaves and fruits are frequently affected by fungal, bacterial, and viral diseases, which can reduce yield and fruit quality. Strawberry, on the other hand, is a high-value crop commonly cultivated in greenhouse environments, where disease detection, fruit counting, quality assessment, yield prediction, and harvesting support are important for production management [19]. Therefore, these crops represent relevant case studies for the application of artificial intelligence and computer vision in modern agriculture.

Plant health monitoring is essential in controlled cultivation systems because biotic and abiotic stresses can strongly affect crop quality and productivity. In lettuce, nutrient imbalance, water stress, inappropriate light conditions, and microbial infections can negatively influence leaf development. These problems are often reflected through changes in leaf color, texture, and shape. Concepcion et al. [20] showed that visual symptoms, such as necrotic and chlorotic regions, can be computationally assessed using image-based indices. In another work, Concepcion et al. [21] proposed a visible-band greenness index to estimate plant growth signatures at the canopy scale, confirming the relevance of image processing for plant health monitoring. Similar visual symptoms can also be observed in tomato and strawberry plants, where disease symptoms may appear as spots, discoloration, wilting, lesions, mold, or abnormal tissue development.

In hydroponically grown lettuce, diseases are mainly associated with bacterial and fungal infections. Koohakan et al. [22] investigated the major diseases affecting lettuce cultivated using the Nutrient Film Technique and reported that bacterial diseases are often related to poor sanitary conditions, plant wounds, and contamination during cultivation practices. These diseases may cause lesions, tissue decay, and severe degradation of the plant structure. Fungal diseases, such as downy mildew, are also common in lettuce and usually develop under cool and humid conditions [23]. They can lead to yellowing, wilting, browning, and a progressive decline in crop quality. Since early disease symptoms are sometimes difficult to detect visually, manual inspection may not be sufficient for accurate and timely diagnosis.

Traditional disease monitoring methods rely mainly on visual inspection and expert knowledge. However, these approaches are time-consuming, subjective, and often inaccurate, particularly when symptoms are at an early stage or when the disease appears in small regions of the leaf. Moreover, continuous expert supervision is not always available in practical agricultural systems. Recent advances in artificial intelligence, computer vision, and image processing have therefore provided new opportunities for automatic plant disease detection. Vishnoi et al. [24] reviewed several computational intelligence and image

processing techniques used for plant disease detection and emphasized their importance in improving diagnosis accuracy. In a later study, Vishnoi et al. [25] presented a comprehensive review of feature extraction techniques for plant leaf disease detection, showing that visual features such as color, texture, and shape play an important role in disease classification. More recent review studies also confirmed that machine learning and deep learning are increasingly used for plant disease classification, pest detection, early diagnosis, and precision agriculture applications [26–28].

To overcome the limitations of handcrafted feature extraction, deep learning approaches, particularly Convolutional Neural Networks, have been widely adopted for plant disease classification. CNNs are able to automatically learn hierarchical features from images, making them suitable for complex visual recognition tasks. The development of deep architectures such as VGG [29], ResNet [30], MobileNet [31], Inception, DenseNet, and EfficientNet has significantly improved image classification performance in various domains. These architectures have also been adapted to agricultural applications, where they are used for detecting diseases, identifying nutrient deficiencies, monitoring crop growth, estimating yield, and supporting decision-making in controlled-environment agriculture.

Several studies have specifically investigated disease and stress detection in hydroponic lettuce. Deng et al. [32] proposed a Support Vector Machine-based method for recognizing disease features in hydroponic lettuce images. Their approach focused on the classification of lettuce leaf diseases using visual features and achieved promising results. Aquino et al. [33] also used an SVM classifier to distinguish between healthy and chlorotic lettuce leaves in a rotating hydroponic system. Their results showed that classical machine learning can be effective when the number of classes is limited and the image acquisition conditions are controlled. However, these methods may face limitations when dealing with complex backgrounds, larger datasets, or subtle disease symptoms.

Deep learning has also been explored for detecting unhealthy lettuce in hydroponic systems. Pratama et al. [34] used a CNN-based object detection approach relying on Faster R-CNN with Inception V2 to assess unhealthy hydroponic lettuce. Their study demonstrated the feasibility of using deep learning for automatic lettuce health monitoring, although the reported performance indicates that further improvements are still needed. In addition, Bryan et al. [35] developed a lettuce root development and monitoring system using machine learning in hydroponics, showing that artificial intelligence can be applied not only to leaf disease detection but also to root growth analysis and crop monitoring.

More recently, Rathor et al. [36] proposed a novel deep convolutional neural network, named Conv-7 DCNN, for the efficient classification of lettuce diseases in hydroponic vertical farming systems. The proposed architecture consists of seven convolutional and pooling layers followed by three fully connected layers. The model was trained using a publicly available Kaggle dataset containing 2,569 RGB images of hydroponically grown lettuce leaves divided into three classes: bacterial disease, fungal disease, and healthy leaves. To improve generalization and reduce overfitting, the authors applied several techniques, including data augmentation, batch normalization, L2 regularization, and a dropout rate of 0.2. The model achieved strong classification performance, with an accuracy, precision, recall, and F1-score of 97.

In addition to disease detection, nutrient deficiency diagnosis has attracted significant attention in hydroponic cultivation. Zhang et al. [37] proposed a Random Forest-based method for diagnosing nutrient deficiencies in whole hydroponic lettuce plants. Their method considered deficiencies related to nitrogen, potassium, and calcium, and used features extracted from segmented plant images. The results showed that Random Forest outperformed other models such as SVM and back-propagation neural networks. Gul and Bora [38] investigated the detection of nutrient deficiencies in hydroponic basil using pre-trained CNN models. Their work showed that transfer learning can be effective for identifying nutrient stress,

while Grad-CAM visualization helped interpret the regions used by the model for prediction. Similarly, Xu et al. [39] applied deep convolutional neural networks for image-based diagnosis of nutrient deficiencies in rice, with DenseNet121 achieving high classification accuracy.

Tomato disease detection has received considerable attention in the field of deep learning-based agriculture. Tomato plants are commonly affected by several diseases, including early blight, late blight, bacterial spot, mosaic virus, leaf mold, and septoria leaf spot. These diseases can affect leaves, stems, and fruits, causing significant yield losses. Anandhakrishnan et al. [40] proposed a deep convolutional neural network for tomato leaf disease identification using image data. Their work showed that CNN-based feature learning can provide effective classification performance compared with traditional approaches. Rehana et al. [41] proposed a region-based convolutional neural network for tomato disease detection, emphasizing the usefulness of region-based approaches for localizing infected areas. Thalor et al. [42] compared CNN models for tomato disease detection under both controlled and real-world image conditions, highlighting the importance of evaluating models beyond laboratory datasets.

More recent works have focused on improving tomato disease detection under complex greenhouse and field conditions. Liu and Wang [43] proposed a tomato disease object detection method combining prior knowledge attention mechanisms and multi-scale features. Their approach aimed to improve disease detection under complex backgrounds, especially when diseased regions are small, dense, or partially occluded. Wang et al. [44] introduced TomatoDet, an efficient deep learning model for tomato disease detection in greenhouse environments. The model incorporated improved feature extraction and feature fusion modules to strengthen the detection of small and occluded disease regions, making it suitable for real-time disease monitoring. Jelali et al. [45] reviewed deep learning networks for tomato disease and pest detection and showed that CNN-based classifiers, object detection models, and lightweight architectures are increasingly used for tomato health monitoring. These studies show that tomato is one of the most important benchmark crops for evaluating deep learning models in plant disease detection.

Strawberry cultivation has also benefited from deep learning-based monitoring systems. Yang and Kim [19] reviewed recent advances in greenhouse strawberry cultivation using deep learning techniques. Their study showed that deep learning has been applied to several strawberry-related tasks, including disease identification, fruit counting, quality assessment, yield prediction, crop management, and harvesting support. For strawberry disease detection, Afzaal et al. [46] proposed an instance segmentation model based on Mask R-CNN. Their work focused on detecting and segmenting multiple strawberry disease categories, demonstrating that instance segmentation can provide more detailed information than simple image classification by localizing the diseased regions. The strawberry disease dataset introduced by Afzaal et al. [47] also provides annotated images for several disease categories, which supports the development of segmentation-based models.

Transfer learning has also been widely used for strawberry disease diagnosis. Karki et al. [48] investigated strawberry disease detection using transfer learning with deep convolutional neural networks. Their study showed that pre-trained CNN models can improve classification performance while reducing the need for very large training datasets. This is particularly important in agriculture, where annotated images are often limited and expensive to collect. In addition, deep learning techniques have been applied to strawberry fruit detection, maturity estimation, and quality grading, confirming the potential of computer vision for greenhouse strawberry production [19]. These works demonstrate that strawberry is a relevant crop for studying disease detection, segmentation, quality control, and intelligent harvesting.

Energy efficiency and real-time deployment are important issues in smart hydroponic

and greenhouse systems. Musa et al. [49] proposed a low-power deep learning model for plant disease detection using knowledge distillation techniques. Their system reduced energy consumption while maintaining good detection performance, making it suitable for smart hydroponic applications where embedded devices and limited power resources are often used. Hati and Singh [50] introduced an AI-driven plant phenotyping model for soilless farming datasets, addressing tasks such as growth analysis, yield stage identification, and plant health monitoring. Recent works have also emphasized the use of lightweight deep learning, edge AI, and mobile-based applications for real-time plant disease detection in smart agriculture.

Beyond hydroponic systems, CNN-based models have achieved high performance in plant disease recognition for several crops. Lee et al. [51] applied a CNN model for potato leaf disease detection and reported high classification accuracy. Amara et al. [52] proposed a deep learning-based method for banana leaf disease classification under challenging image conditions. Waheed et al. [53] developed an optimized DenseNet model for corn leaf disease recognition and classification. Vishnoi et al. [54] also used CNNs for detecting apple plant diseases from leaf images. These studies demonstrate the strong potential of deep learning for agricultural disease diagnosis, although model performance depends heavily on dataset quality, image diversity, and training strategy.

Datasets play a central role in the development of reliable deep learning models. Public datasets, such as PlantVillage, PlantDoc, and crop-specific datasets, provide useful resources for training and evaluating classification models. In the case of lettuce, public datasets such as the lettuce plant disease dataset available on Kaggle [55] can support model development and benchmarking. For strawberry, the dataset proposed by Afzaal et al. [47] provides images and annotations for segmentation tasks. However, many agricultural datasets remain limited in terms of size, disease diversity, environmental variability, and real-world acquisition conditions. As a result, data augmentation, transfer learning, model optimization, and cross-domain validation are commonly used to improve generalization. Kandel and Castelli [56] showed that batch size can influence the generalizability of CNN models, which is an important consideration when training deep learning models on limited datasets.

Overall, the reviewed literature shows that artificial intelligence, computer vision, and deep learning have become powerful tools for plant disease detection and crop monitoring. In hydroponic and greenhouse systems, these technologies can help identify diseases, detect nutrient deficiencies, monitor plant growth, estimate yield, and support early intervention. Studies on lettuce, tomato, and strawberry confirm that CNN-based models, transfer learning, object detection, instance segmentation, attention mechanisms, and lightweight models can improve the automatic recognition of plant health problems. However, several challenges remain, including the limited availability of annotated datasets, the detection of early or small symptoms, robustness under real cultivation conditions, and deployment on low-power devices. Therefore, further research is required to develop efficient and reliable deep learning models for real-time monitoring of plant health in controlled-environment agriculture.

As shown in Figure 1.18, the reviewed literature has evolved significantly over time. Early studies mainly focused on general plant disease detection and basic hydroponic disease identification, while recent works increasingly address crop-specific applications such as lettuce, tomato, and strawberry monitoring. The year 2022 shows a strong concentration of studies related to hydroponic systems and lettuce disease detection, whereas 2023 and 2024 are marked by a growing interest in tomato and strawberry disease detection using advanced deep learning models. The figure also shows that fewer studies focus on edge deployment and integrated smart systems, which highlights the originality of the proposed VertiFarm platform. By combining hydroponic monitoring, fuzzy logic control, disease detection, crop status assessment, and harvest-oriented decision support, the present work addresses a gap

that remains insufficiently covered in the existing literature.

Based on the reviewed studies, it can be concluded that most existing works address plant disease detection, nutrient deficiency diagnosis, or hydroponic monitoring as separate tasks. In contrast, the present work contributes to the development of VertiFarm, an integrated hydroponic Digital Twin designed to monitor, control, diagnose, and support crop production in a unified edge-based platform. The proposed system combines real-time IoT sensing, historical data storage, fuzzy logic-based chemical regulation, deep learning-based disease detection, and a dashboard-assisted operator interface. A key contribution of this work is its ability to support crop status assessment and harvest estimation by exploiting multiple sources of information, including pH, EC, temperature, humidity, crop profile, historical trends, and visual plant health indicators. By detecting diseases in lettuce, tomato, and strawberry plants and linking these diagnoses with the recorded evolution of the cultivation environment, VertiFarm provides useful decision-support information for evaluating plant development and anticipating the expected harvest condition. Therefore, the proposed platform moves beyond simple monitoring by offering an intelligent and adaptive system capable of improving crop management, reducing production losses, and supporting harvest-oriented decision-making in hydroponic vertical farming.

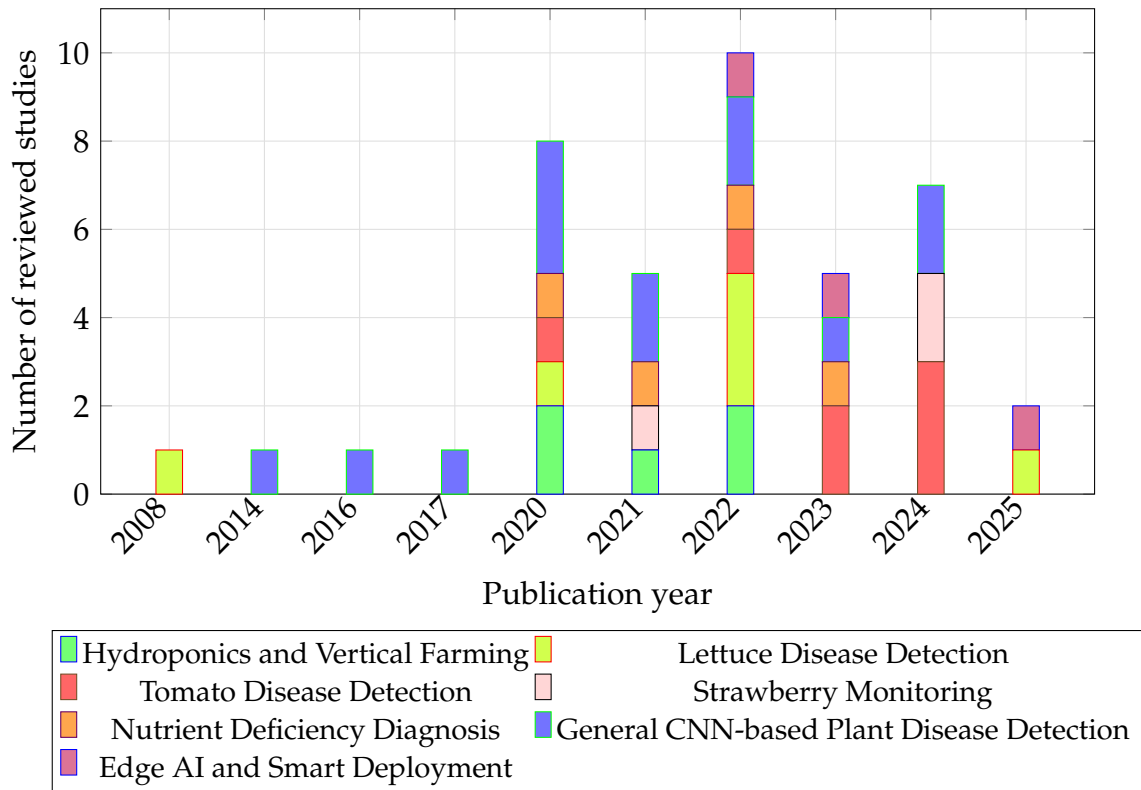


Figure 1.18: Distribution of reviewed studies by publication year and research domain.

## **Part 2: Fuzzy Logic**

Before introducing fuzzy logic, it is important to place it within the broader field of Artificial Intelligence (AI). AI aims to develop systems capable of reasoning, learning, and making decisions in a way that imitates human intelligence. However, real-world problems are often uncertain, imprecise, or difficult to describe using only binary logic, where a statement is either true or false. For this reason, fuzzy logic has emerged as an important AI approach that allows machines to reason with approximate values and linguistic concepts such as “high”, “low”, “medium”, or “acceptable”. Therefore, fuzzy logic provides a flexible way to model human reasoning and decision-making when information is incomplete or not clearly defined.

Fuzzy logic is a mathematical framework that extends classical binary logic by allowing degrees of truth rather than strict true/false values. Its ability to handle uncertainty and imprecision makes it well-suited for real-world agricultural applications.

## 7.1 Definition and Basic Concepts

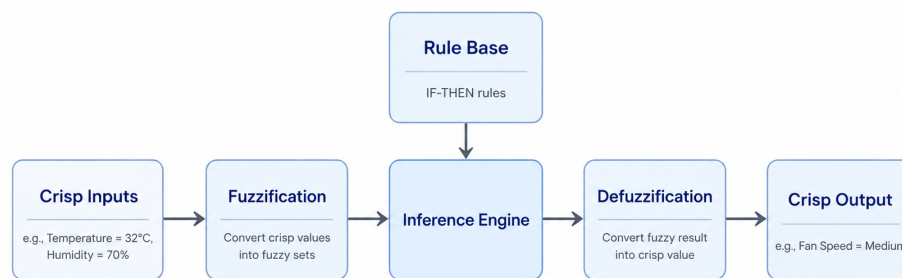


Figure 1.19: fuzzy logic

A fuzzy logic system, as illustrated in Figure 1.19, is a rule-based inference framework that handles imprecise or uncertain information by mimicking human reasoning. Its architecture is composed of four main components:

- **Crisp Inputs:** The system receives precise, measurable values from the environment (e.g., Temperature = 32°C, Humidity = 70%), which serve as the entry point of the inference process.
- **Fuzzification:** the crisp input values are converted into fuzzy sets using membership functions, assigning a degree of membership (between 0 and 1) to each linguistic variable (e.g., *hot*, *medium*, *cold*).
- **Inference Engine:** guided by a **Rule Base** containing a set of IF-THEN linguistic rules (e.g., *IF temperature is high AND humidity is high THEN fan speed is fast*), the inference engine evaluates the activated rules and computes the corresponding fuzzy output.
- **Defuzzification:** the resulting fuzzy output is converted back into a single crisp value using methods such as the centroid or the weighted average, producing a concrete, actionable decision.

- **Crisp Output:** the final decision is expressed as a precise value (e.g., Fan Speed = Medium), ready to be used by the controlled system.

In this work, fuzzy logic was integrated into the system to handle the inherent uncertainty and variability of agricultural sensor data, enabling more robust and interpretable decision-making under real-world conditions.

## 7.2 Difference Between Classical Logic and Fuzzy Logic

Classical logic and fuzzy logic differ fundamentally in how they represent and process truth values. While classical logic relies on binary values, fuzzy logic allows gradual assessment of truth.

Table 1.4: Comparison between Classical Logic and Fuzzy Logic

| Criteria             | Classical Logic               | Fuzzy Logic                     |
|----------------------|-------------------------------|---------------------------------|
| Truth Values         | Binary (0 or 1)               | Continuous (between 0 and 1)    |
| Nature               | Crisp and precise             | Approximate and flexible        |
| Set Membership       | Element either belongs or not | Partial membership allowed      |
| Handling Uncertainty | Not suitable                  | Designed to handle uncertainty  |
| Logic Type           | Boolean logic                 | Multi-valued logic              |
| Decision Making      | Rigid                         | Gradual and adaptive            |
| Applications         | Digital circuits, mathematics | Control systems, AI, automation |

As shown in Table 1.4, fuzzy logic provides a more realistic approach when dealing with real-world problems where uncertainty and imprecision are present. Unlike classical logic, which forces strict true/false decisions, fuzzy logic allows intermediate values, making it particularly useful in intelligent systems and control applications.

## Conclusion

This chapter provided a general theoretical background on the main Artificial Intelligence techniques used in smart agriculture. It introduced machine learning and deep learning as important approaches for data analysis and prediction, with a particular focus on Convolutional Neural Networks for image-based applications. CNN architectures, transfer learning, ResNet, and EfficientNet-B3 were presented as key elements for understanding modern image classification systems.

The chapter also explained the transition from simple image classification to more advanced computer vision tasks such as object detection and segmentation. In this context, YOLO was introduced as a powerful real-time object detection algorithm, and its architecture was described through its main components: the backbone, the neck, and the head. Evaluation metrics such as accuracy, precision, recall, F1-score, IoU, and mAP were also presented to show how model performance can be measured.

In addition, fuzzy logic was discussed as a flexible reasoning method capable of dealing with uncertain and imprecise information. This is particularly relevant in agricultural systems, where sensor data and environmental conditions are often variable. Overall, the concepts presented in this chapter form the theoretical basis for the design and development of the proposed intelligent system. The next chapter will focus on the methodology, system architecture, and implementation steps adopted in this work.

## DESIGN AND MODELING OF PROPOSED AI SOLUTIONS

## Introduction

Building upon the theoretical foundations established in the preceding chapter, this chapter presents the practical implementation of the artificial intelligence models developed within the proposed system. Two principal models are addressed. The first concerns plant leaf disease detection, for which three deep learning architectures were designed, trained, and compared under identical experimental conditions: a **Custom Convolutional Neural Network** trained from scratch, **ResNet-50**, and **EfficientNet-B3**, both fine-tuned from ImageNet pretrained weights. A thorough comparison between these three architectures is conducted in order to identify the most suitable model for deployment in a hydroponic monitoring context. The second model addresses crop yield estimation, achieved through an image segmentation approach that enables the quantitative analysis of plant growth and the prediction of harvest output.

## 1 Plant Disease Detection Models

This part addresses the detection of plant leaf diseases using deep convolutional neural network models. Since all three architectures — a **Custom Convolutional Neural Network** trained from scratch, **ResNet-50**, and **EfficientNet-B3**, both fine-tuned from ImageNet pretrained weights — operate on the same data, the discussion begins with a unified presentation of the dataset employed, consisting of a balanced corpus of 23,212 labelled images spanning 20 categories (18 disease classes and 2 healthy classes) across three plant species. This is followed by a description of the training process and experimental protocol applied to each model, before presenting the results obtained and comparing the performance of the three architectures.

### 1.1 Dataset Acquisition and Processing

No single publicly available dataset covers the three hydroponic crops of interest — tomato, strawberry, and lettuce — with sufficient class diversity and volume for deep learning. To address this, three curated datasets were selected, individually analysed, then merged into a single unified corpus. The merging process involved rigorous preprocessing at every stage: cleaning and filtering each source, harmonising incompatible label conventions, balancing class sizes through downsampling and augmentation, and finally splitting the corpus using a stratified 80/10/10 strategy. Each methodological decision is justified in detail, as the quality of the training data directly caps the achievable model accuracy.

## Description of Source Datasets

Three datasets were selected according to four criteria: their relevance to hydroponic crop species, sufficient class diversity for multi-class classification, public availability with documented provenance, and acceptable baseline image quality.

Table 2.1: Summary of the three source datasets prior to preprocessing.

| Dataset                           | Plant           | Classes   | Images         | Resolution   | Acquisition                                |
|-----------------------------------|-----------------|-----------|----------------|--------------|--------------------------------------------|
| PlantVillage [57]                 | Tomato (subset) | 10        | 18 160         | 256 × 256 px | Controlled laboratory (uniform background) |
| Strawberry Disease Detection [58] | Strawberry      | 7         | 2 500          | 419 × 419 px | Real greenhouse environment                |
| Lettuce Plant Disease [59]        | Lettuce         | 3         | 2 813          | Mixed        | Real field environment                     |
| <b>Total (raw)</b>                | <b>3 plants</b> | <b>20</b> | <b>23 473*</b> | —            | Mixed                                      |

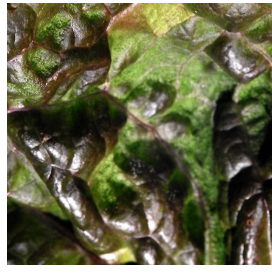
\* 261 images removed during cleaning, yielding the 23 212-image corpus used in all subsequent experiments (Table 2.4).

## Dataset Samples

Figure 2.1 presents representative samples of diseased plant leaves drawn from the three crop categories included in our dataset: tomato, lettuce, and strawberry. Each image corresponds to a distinct disease class used during model training.



(a) Lettuce – Downy Mildew



(b) Tomato – Early Blight



(c) Lettuce – Bacterial Leaf Spot



(d) Tomato – Mosaic Virus



(e) Tomato – Late Blight



(f) Strawberry – Leaf Spot



(g) Strawberry – Angular Leaf Spot



(h) Strawberry – Powdery\_mildew

Figure 2.1: Sample images from the dataset illustrating various plant disease classes across tomato, lettuce, and strawberry.

## Dataset Preprocessing

Prior to merging, each dataset underwent a targeted cleaning step: from **PlantVillage**, only the 10 tomato-specific classes were retained; from the **Strawberry** dataset, all JSON annotation files and the unlabelled severity folder were removed; and the **Lettuce** dataset was reorganised into per-class collections. All class labels were then normalised to a unified **Plant\_\_\_Disease** format, keeping diseases distinct across plant species to preserve their visual specificity. To address class imbalance, two complementary strategies were applied: the *Tomato Yellow Leaf Curl Virus* class (5 357 images) was randomly downsampled to 1 500 images (seed = 42), reducing the imbalance ratio from  $R = 14.37$  to  $R = 5.71$ ; and nine classes falling below the 800-image threshold were augmented using the Albumentations library [60] until each reached exactly 800 images, through the pipeline detailed in Table 2.2. Augmented images were stored alongside their originals to ensure proportional distribution across all splits, while transformations incompatible with disease classification — such as CutMix, MixUp, aggressive cropping, and grayscale conversion — were deliberately excluded.

Table 2.2: Augmentation pipeline applied to minority classes, with biological and technical justification.

| Transformation             | Probability Justification |                                                          |
|----------------------------|---------------------------|----------------------------------------------------------|
| Horizontal flip            | $p = 0.50$                | Leaves have no fixed orientation in hydroponic imaging.  |
| Vertical flip              | $p = 0.30$                | Valid for overhead greenhouse camera geometry.           |
| 90° rotation               | $p = 0.50$                | Improves rotational invariance for varied camera angles. |
| Brightness/contrast jitter | $p = 0.60$                | Handles lighting variation across datasets.              |
| Hue/saturation/value shift | $p = 0.50$                | Reduces camera color bias differences.                   |
| Gaussian blur              | $p = 0.30$                | Simulates slight defocus in leaf edges.                  |
| Grid distortion            | $p = 0.30$                | Models natural leaf deformation.                         |
| Coarse dropout             | $p = 0.30$                | Improves robustness to missing symptom regions.          |

## Data Preprocessing Pipeline

Table 2.3: Preprocessing and online augmentation pipeline.

| Stage                            | Operation                             | Details                                                                                 |
|----------------------------------|---------------------------------------|-----------------------------------------------------------------------------------------|
| Resizing (Standard)              | 224×224 px                            | Bicubic interpolation                                                                   |
| Resizing (EfficientNetB3)        | 300×300 px                            | Matches pretrained input size                                                           |
| Normalization                    | ImageNet stats                        | mean [0.485, 0.456, 0.406], std [0.229, 0.224, 0.225]                                   |
| Online augmentation (train only) | Flip/rotate/jitter                    | Flip ( $p=0.5$ ), rotation ( $\pm 15^\circ$ ), color jitter ( $\pm 20$ Validation/Test) |
| None                             | Only resizing + normalization applied |                                                                                         |

All images were resized to **224 × 224 px** (300 × 300 for EfficientNetB3) and normalised to the ImageNet distribution. During training, online augmentation was applied to improve generalisation, including flips, rotation, and colour jitter. Validation and test images received only resizing and normalisation to ensure reproducible evaluation. Table 2.3 summarises the full pipeline.

## Data Splitting Strategy

The unified corpus was partitioned into training, validation, and test subsets using a **stratified random split** (80/10/10, seed = 42), ensuring that the class distribution is preserved across all three subsets. The split was performed at the corpus level rather than inheriting the original per-dataset splits, which were inconsistent across sources. Table 2.4 presents the final per-class image counts.

## Integration Pipeline Architecture

Figure 2.2 illustrates the five-stage data integration pipeline implemented within a Kaggle Notebook environment. The pipeline begins with the validation of three read-only source datasets, followed by an exploratory analysis of per-class distributions. In Stage 3, images from all sources are merged into a unified directory structure conforming to a common label namespace. Stage 4 applies class balancing through targeted downsampling and

Table 2.4: Final dataset distribution after augmentation and stratified 80/10/10 split.

| Subset       | Images        | Percentage  | Notes                         |
|--------------|---------------|-------------|-------------------------------|
| Training     | 18,565        | 80%         | Stratified split<br>Seed = 42 |
| Validation   | 2,321         | 10%         |                               |
| Test         | 2,326         | 10%         | Final evaluation set          |
| <b>Total</b> | <b>23,212</b> | <b>100%</b> | After augmentation            |

Albumentations-based augmentation, ensuring a minimum of 800 images per class. Finally, Stage 5 performs a stratified three-way split (80/10/10) using a fixed random seed (42), producing the train/, val/, and test/ subsets used throughout all subsequent experiments.

## 1.2 Rationale for Architecture Selection

Three models were selected to cover a representative spectrum of complexity, pretraining, and computational cost, while remaining widely adopted in the plant disease detection literature.

The **Custom CNN** serves as the methodological baseline, trained from scratch to quantify empirically the contribution of pretraining. **ResNet-50** represents the canonical residual network: its identity shortcut connections overcome the vanishing gradient problem, and its prevalence in published benchmarks [6] enables direct comparison. **EfficientNet-B3** [61] represents the compound-scaling paradigm, jointly optimising depth, width, and resolution; the B3 variant was retained as the optimal trade-off between accuracy and computational cost. Table 2.5 summarises the key properties of all three models.

Table 2.5: Architectural properties of the three evaluated models.

| Property                 | Custom CNN          | ResNet-50             | EfficientNet-B3     |
|--------------------------|---------------------|-----------------------|---------------------|
| Pretrained weights       | None (scratch)      | ImageNet              | ImageNet            |
| Total parameters         | 7.98M               | 25.6M                 | 10.7M               |
| Input resolution         | 224×224             | 224×224               | 300×300             |
| Architectural paradigm   | Custom residual CNN | Deep residual network | Compound-scaled CNN |
| Published ImageNet top-1 | N/A                 | 76.1%                 | 82.0%               |
| Literature prevalence    | Baseline            | High [6]              | High [62]           |

## 1.3 Training Environment

The training and evaluation experiments were conducted using a Kaggle Notebook environment equipped with an NVIDIA Tesla T4 GPU featuring 15.6 GB of VRAM. The software environment was based on Python 3.12 and PyTorch 2.x, with additional libraries including torchvision, scikit-learn, and Albumentations. Data augmentation was performed using a hybrid strategy that combined offline augmentation through Albumentations with

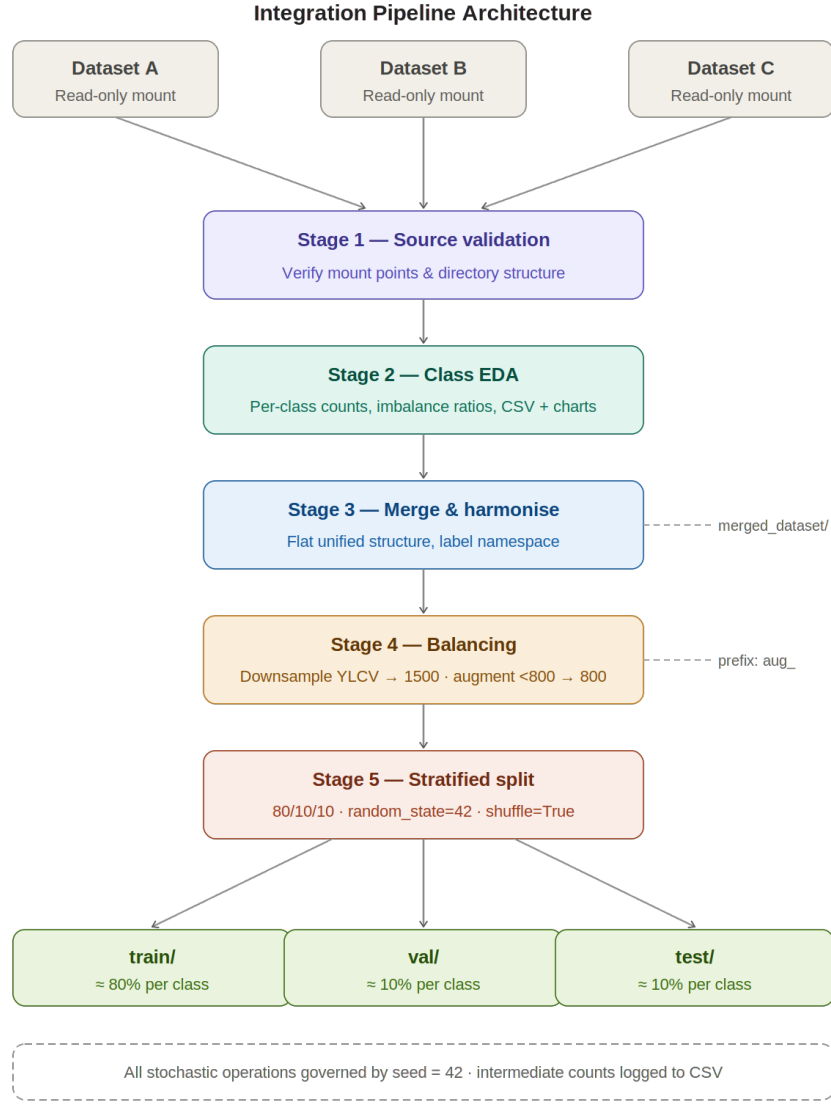


Figure 2.2: Integration pipeline architecture: from raw source datasets to the final stratified train/val/test split.

online augmentation implemented via `transforms.Compose`. To ensure experiment reproducibility, a fixed random seed of 42 was applied to all stochastic operations. Dataset loading was handled using the `ImageFolder` class, which automatically assigns class indices in alphabetical order. The `DataLoader` was configured with a batch size of 32, two worker processes (`num_workers=2`), and pinned memory (`pin_memory=True`) to improve data transfer efficiency between the host memory and the GPU.

## 1.4 Custom Convolutional Neural Network

A custom CNN was designed from the ground up for the 20-class hydroponic leaf disease classification task. The architecture incorporates a set of modern design principles that collectively enable stable training from random initialisation on a dataset of moderate size, without recourse to pretrained feature representations.

The network is organised into five sequential stages preceded by a stem block, followed by a global pooling head and a two-layer classifier. Table 2.6 describes the complete archi-

texture with output dimensions at each stage.

Table 2.6: Custom CNN architecture — layer sequence and output feature map dimensions (input:  $3 \times 224 \times 224$ ).

| Stage   | Operations                                                                                                                                                        | Output shape              | Design rationale                                     |
|---------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------|------------------------------------------------------|
| Input   | —                                                                                                                                                                 | $3 \times 224 \times 224$ | —                                                    |
| Stem    | Conv $7 \times 7$ , 32 filters, stride 2 $\rightarrow$ BN $\rightarrow$ ReLU $\rightarrow$ MaxPool $2 \times 2$                                                   | $32 \times 56 \times 56$  | Large receptive field captures global leaf structure |
| Stage 1 | ConvBlock( $32 \rightarrow 64$ ) + ResBlock(64) $\rightarrow$ MaxPool                                                                                             | $64 \times 28 \times 28$  | Low-level edge and colour features                   |
| Stage 2 | ConvBlock( $64 \rightarrow 128$ ) + ResBlock(128) $\rightarrow$ MaxPool                                                                                           | $128 \times 14 \times 14$ | Mid-level lesion shape features                      |
| Stage 3 | ConvBlock( $128 \rightarrow 256$ ) + ResBlock(256) $\rightarrow$ MaxPool                                                                                          | $256 \times 7 \times 7$   | High-level disease morphology                        |
| Stage 4 | ConvBlock( $256 \rightarrow 512$ ) + ResBlock(512) + SpatialDropout(0.1)                                                                                          | $512 \times 7 \times 7$   | Deep disease signatures                              |
| Head    | GAP $\rightarrow$ Dropout(0.4) $\rightarrow$ Linear(512, 256) $\rightarrow$ LayerNorm $\rightarrow$ ReLU $\rightarrow$ Dropout(0.3) $\rightarrow$ Linear(256, 20) | 20                        | Classification logits                                |

## 1.5 ResNet-50

The ResNet-50 model was loaded with ImageNet pretrained weights (ResNet50\_Weights.DEFAULT) and adapted for the 20-class classification task by replacing the original Linear(2,048, 1,000) classification layer with a task-specific head comprising a Dropout layer ( $p = 0.4$ ) followed by a Linear(2,048, 20) layer.

Training was structured into two sequential phases to prevent catastrophic forgetting of the pretrained feature representations:

**Phase A (head-only training, 5 epochs).** All backbone parameters were frozen (`requires_grad = False`). Only the newly initialised classification head was optimised, using the Adam optimiser with an initial learning rate of  $10^{-3}$ , decayed to  $10^{-5}$  via cosine annealing. This phase establishes a stable and well-conditioned mapping from ResNet-50's 2,048-dimensional feature space to the 20-class label space, ensuring that the gradients propagating back into the backbone during Phase B originate from a reliable classifier rather than a randomly initialised one.

**Phase B (fine-tuning, 15 epochs).** The final residual stage (`model.layer4`) was unfrozen, in addition to the classification head. The learning rate was reduced to  $10^{-4}$ , decayed to  $10^{-6}$  via cosine annealing. The final residual stage encodes the most abstract, task-specific

feature representations — the spatial patterns corresponding to lesion morphology, symptom colouration, and disease texture — which require domain-specific adaptation for leaf disease classification. The three preceding stages (layer1–layer3) were retained frozen, as their lower-level representations (edges, colour gradients, texture boundaries) are sufficiently general-purpose to transfer without modification.

## 1.6 EfficientNet-B3

EfficientNet-B3 was loaded with ImageNet pretrained weights (`EfficientNet_B3_Weights.DEFAULT`) and adapted by replacing the original `model.classifier` module with a task-specific head:  $\text{Dropout}(p=0.4) \rightarrow \text{Linear}(1,536, 20)$ , where 1,536 is the output dimensionality of the EfficientNet-B3 backbone’s final convolutional stage.

The same two-phase transfer learning strategy as ResNet-50 was applied. In Phase B, the final three feature blocks of the backbone (`model.features[6], [7], [8]`) were unfrozen, increasing the number of trainable parameters from 30,740 (head only) to 8,536,818 (head and top three blocks). The input resolution was set to  $300 \times 300$  pixels, consistent with the pretraining resolution of the B3 variant, as specified in the data preprocessing pipeline (Table 2.3, Section 1.1).

The difference in fine-tuning depth between ResNet-50 (one stage unfrozen) and EfficientNet-B3 (three blocks unfrozen) reflects their distinct architectural scales. ResNet-50’s layer4 constitutes approximately 40 % of the total backbone parameters and encodes the highest-level task-specific features. In contrast, EfficientNet-B3’s individual MBConv blocks each contain fewer parameters, so unfreezing three blocks achieves a comparable proportion of trainable backbone capacity. Both choices were validated empirically by observing convergence and validation accuracy during Phase B.

## 1.7 Summary of Hyperparameters

Table 2.7 provides a complete, consolidated reference of all training hyperparameters for the three models.

Table 2.7: Training hyperparameters for all three models.

| Hyperparameter         | Custom CNN                      | ResNet-50        | EfficientNet-B3  |
|------------------------|---------------------------------|------------------|------------------|
| Pretrained weights     | None                            | ImageNet         | ImageNet         |
| Input resolution       | $224 \times 224$                | $224 \times 224$ | $300 \times 300$ |
| Total parameters       | 7.98M                           | 25.6M            | 10.7M            |
| Loss function          | Cross-entropy + label smoothing | Cross-entropy    | Cross-entropy    |
| Optimizer              | AdamW                           | Adam             | Adam             |
| Learning rate strategy | Fixed / cosine                  | Cosine annealing | Cosine annealing |
| Epochs                 | 50                              | 20               | 20               |
| Batch size             | 32                              |                  |                  |
| Early stopping         | 10 epochs                       | —                | —                |
| Hardware               | NVIDIA Tesla T4 GPU             |                  |                  |
| Framework              | PyTorch 2.x / torchvision       |                  |                  |

## 1.8 CNN Results

### Training Dynamics

Figure 2.3 presents the training and validation accuracy and loss curves over the full 50-epoch run.

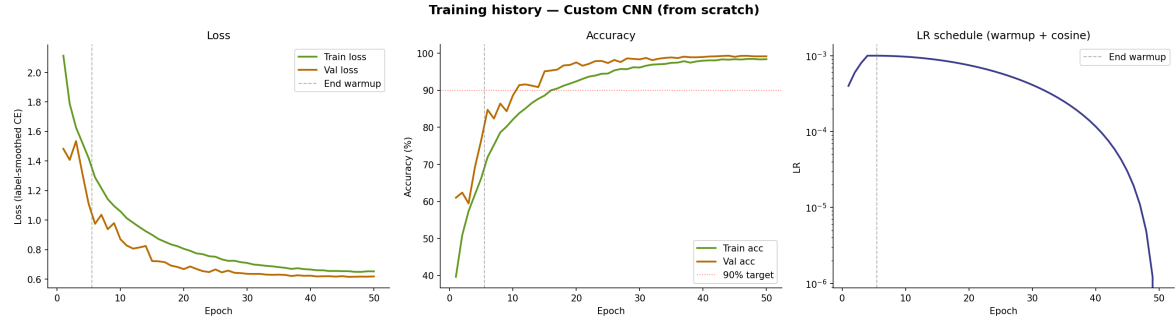
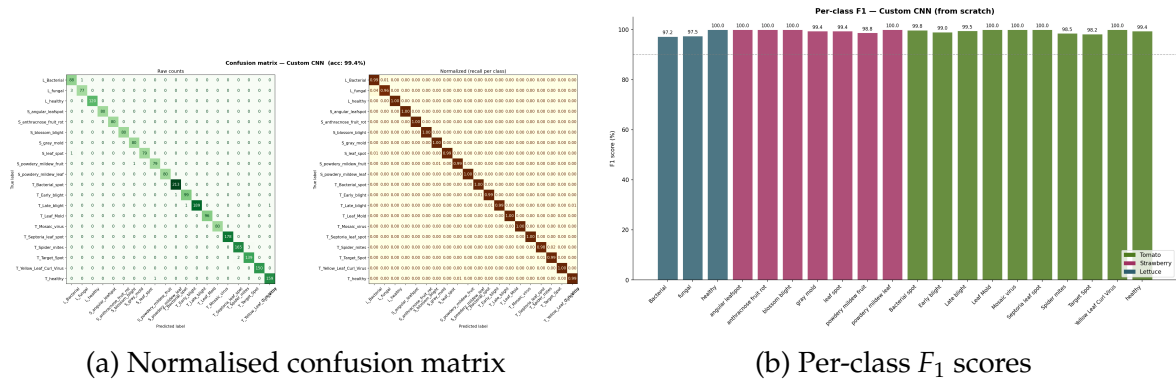


Figure 2.3: CNN training history over 50 epochs.

Three structurally distinct phases are apparent in the learning curves. During the **warmup phase** (epochs 1–5), validation accuracy increased steadily from 61.0% to 76.6% while the learning rate ramped linearly from  $4 \times 10^{-4}$  to  $10^{-3}$ . The constrained learning rate during this phase prevented the high-variance gradients arising from random weight initialisation from producing destructive early updates. The **fast learning phase** (epochs 6–20) produced rapid improvement from 84.7% to 97.5%, driven by the full learning rate operating in conjunction with well-conditioned backbone features. The consistent improvement during this phase is attributable in part to the residual connections in each stage, which maintained active gradient flow through all layers throughout the extended training run. The **refinement phase** (epochs 21–50) exhibited slower but uninterrupted improvement from 97.5% to 99.35%, facilitated by the cosine decay schedule, which progressively reduced the learning rate to near zero while continuing to improve the model’s decision boundaries on the most ambiguous samples.

### Test Set Performance

The model selected for evaluation corresponds to the checkpoint saved at epoch 44, which achieved the highest validation accuracy of 99.35%. Figure 2.4 presents the confusion matrix and per-class  $F_1$  scores on the 2,326-image test set.



(a) Normalised confusion matrix

(b) Per-class  $F_1$  scores

Figure 2.4: CNN test set evaluation.

Table 2.8 presents the complete per-class classification report.

Table 2.8: CNN per-class precision, recall, and  $F_1$  on the test set (2,326 images, 20 classes).

| Plant                        | Class                   | Precision     | Recall       | $F_1$        | Support |
|------------------------------|-------------------------|---------------|--------------|--------------|---------|
| Lettuce                      | Bacterial               | 0.957         | 0.989        | 0.972        | 89      |
|                              | Fungal                  | 0.987         | 0.963        | 0.975        | 80      |
|                              | Healthy                 | 1.000         | 1.000        | 1.000        | 120     |
| Strawberry                   | Angular leafspot        | 1.000         | 1.000        | 1.000        | 80      |
|                              | Anthraco nose fruit rot | 1.000         | 1.000        | 1.000        | 80      |
|                              | Blossom blight          | 1.000         | 1.000        | 1.000        | 80      |
|                              | Gray mold               | 0.988         | 1.000        | 0.994        | 80      |
|                              | Leaf spot               | 1.000         | 0.988        | 0.994        | 80      |
|                              | Powdery mildew (fruit)  | 0.988         | 0.988        | 0.988        | 80      |
|                              | Powdery mildew (leaf)   | 1.000         | 1.000        | 1.000        | 80      |
| Tomato                       | Bacterial spot          | 0.995         | 1.000        | 0.998        | 213     |
|                              | Early blight            | 0.990         | 0.990        | 0.990        | 100     |
|                              | Late blight             | 1.000         | 0.990        | 0.995        | 191     |
|                              | Leaf mold               | 1.000         | 1.000        | 1.000        | 96      |
|                              | Mosaic virus            | 1.000         | 1.000        | 1.000        | 80      |
|                              | Septoria leaf spot      | 1.000         | 1.000        | 1.000        | 178     |
|                              | Spider mites            | 0.988         | 0.982        | 0.985        | 168     |
|                              | Target spot             | 0.979         | 0.986        | 0.982        | 141     |
|                              | Yellow leaf curl virus  | 1.000         | 1.000        | 1.000        | 150     |
|                              | Healthy                 | 0.994         | 0.994        | 0.994        | 160     |
| <b>Macro average</b>         |                         | <b>0.993</b>  | <b>0.993</b> | <b>0.993</b> | 2326    |
| <b>Weighted average</b>      |                         | <b>0.994</b>  | <b>0.994</b> | <b>0.994</b> | 2326    |
| <b>Overall test accuracy</b> |                         | <b>99.40%</b> |              |              |         |

## 1.9 ResNet-50 Results

Figure 2.5 presents the ResNet-50 training curves over 20 epochs

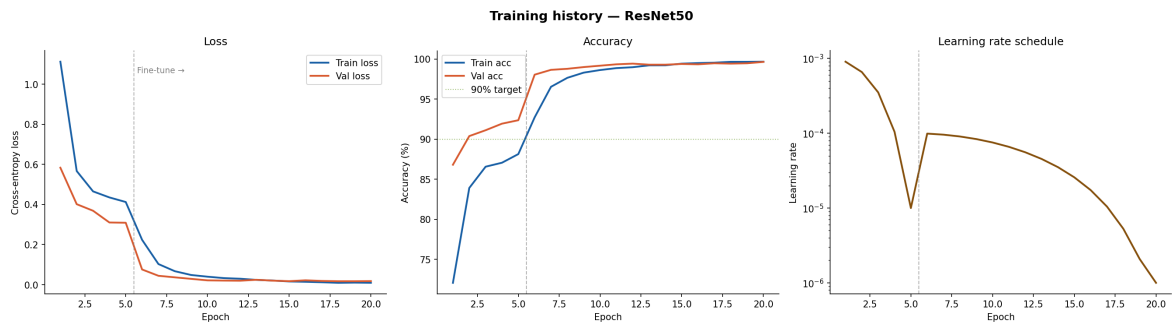


Figure 2.5: ResNet-50 training history over 20 epochs. The vertical dashed line marks the Phase A to Phase B transition.

**Loss:** The train and validation loss curves converge rapidly and consistently, with no sign of overfitting. The transition to fine-tuning (epoch 6) triggers a sharp drop in validation loss, indicating that unfreezing the deeper layers was beneficial.

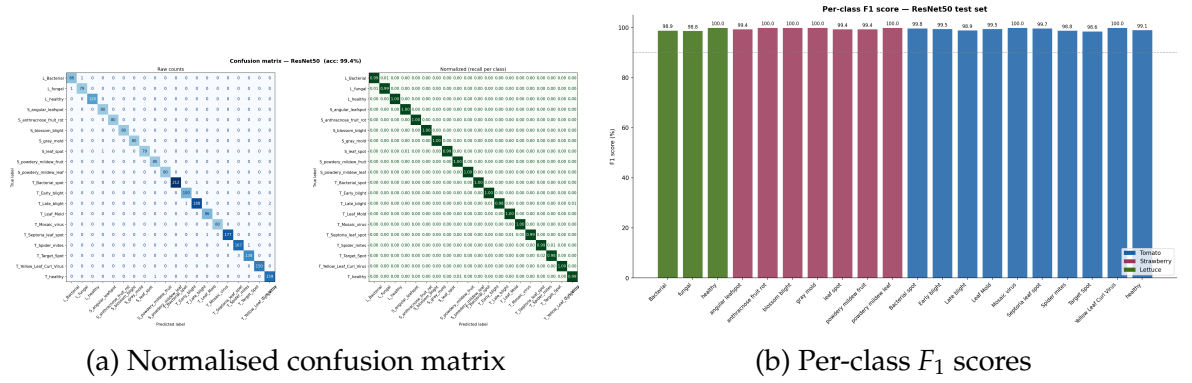
**Accuracy:** The validation accuracy exceeds the training accuracy in the early epochs (due to dropout/augmentation active during training), before both converge toward ~99–

100%. The 90% threshold is reached as early as epoch 7, and the **99% validation threshold is reached at epoch 9**, consistent with the convergence figure reported in Table 2.11.

**Learning rate:** The scheduler applies a strong decay during the feature extraction phase (epochs 1–5), resets to  $10^{-4}$  at the start of fine-tuning, then follows a cosine decay down to  $10^{-6}$ , which explains the stable convergence at the end of training.

## Test Set Performance

The checkpoint selected for evaluation corresponds to the model achieving the highest validation accuracy during the fine-tuning stage. Figure 2.6 presents the normalised confusion matrix and per-class  $F_1$  scores on the 2,326-image test set.



(a) Normalised confusion matrix

(b) Per-class  $F_1$  scores

Figure 2.6: ResNet-50 test set evaluation.

The confusion matrix demonstrates that ResNet-50 achieved highly reliable classification performance across all twenty disease categories. Most classes were classified without error, while the few observed misclassifications occurred between visually similar disease symptoms. The per-class  $F_1$  scores remain consistently high, with all classes exceeding 0.98, confirming that the model maintains balanced performance across the entire dataset and does not favour any particular class.

Table 2.9 presents the complete per-class classification report.

Table 2.9: ResNet-50 per-class precision, recall, and  $F_1$  on the test set (2,326 images, 20 classes).

| Plant                        | Class                  | Precision     | Recall       | $F_1$        | Support |
|------------------------------|------------------------|---------------|--------------|--------------|---------|
| Lettuce                      | Bacterial              | 0.989         | 0.989        | 0.989        | 89      |
|                              | Fungal                 | 0.988         | 0.988        | 0.988        | 80      |
|                              | Healthy                | 1.000         | 1.000        | 1.000        | 120     |
| Strawberry                   | Angular leafspot       | 0.988         | 1.000        | 0.994        | 80      |
|                              | Anthrachnose fruit rot | 1.000         | 1.000        | 1.000        | 80      |
|                              | Blossom blight         | 1.000         | 1.000        | 1.000        | 80      |
|                              | Gray mold              | 1.000         | 1.000        | 1.000        | 80      |
|                              | Leaf spot              | 1.000         | 0.988        | 0.994        | 80      |
|                              | Powdery mildew (fruit) | 0.988         | 1.000        | 0.994        | 80      |
|                              | Powdery mildew (leaf)  | 1.000         | 1.000        | 1.000        | 80      |
| Tomato                       | Bacterial spot         | 1.000         | 0.995        | 0.998        | 213     |
|                              | Early blight           | 0.990         | 1.000        | 0.995        | 100     |
|                              | Late blight            | 0.995         | 0.984        | 0.989        | 191     |
|                              | Leaf mold              | 0.990         | 1.000        | 0.995        | 96      |
|                              | Mosaic virus           | 1.000         | 1.000        | 1.000        | 80      |
|                              | Septoria leaf spot     | 1.000         | 0.994        | 0.997        | 178     |
|                              | Spider mites           | 0.982         | 0.994        | 0.988        | 168     |
|                              | Target spot            | 0.993         | 0.979        | 0.986        | 141     |
|                              | Yellow leaf curl virus | 1.000         | 1.000        | 1.000        | 150     |
|                              | Healthy                | 0.988         | 0.994        | 0.991        | 160     |
| <b>Macro average</b>         |                        | <b>0.994</b>  | <b>0.995</b> | <b>0.995</b> | 2326    |
| <b>Weighted average</b>      |                        | <b>0.994</b>  | <b>0.994</b> | <b>0.994</b> | 2326    |
| <b>Overall test accuracy</b> |                        | <b>99.44%</b> |              |              |         |

## 1.10 EfficientNet-B3 Results

Figure 2.7 presents the EfficientNet-B3 training and validation curves over 20 epochs.

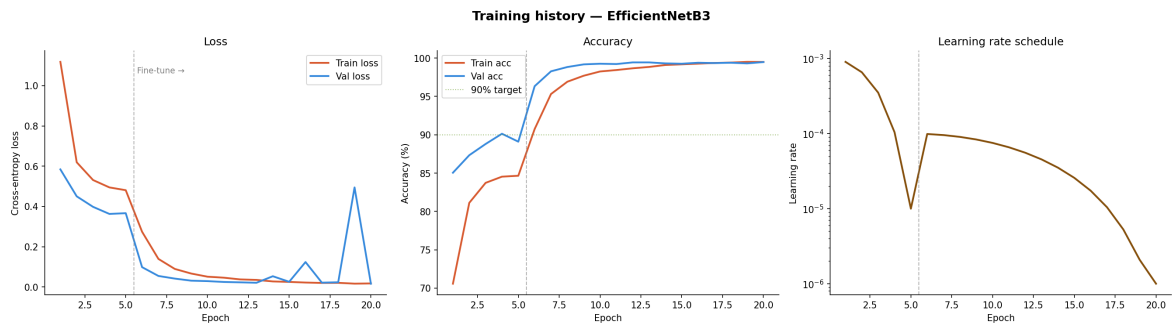


Figure 2.7: EfficientNet-B3 training history over 20 epochs

Phase A (epochs 1–5) improved validation accuracy from 85.05% to 90.13%, demonstrating that the pretrained EfficientNet-B3 feature representations generalise to leaf disease classification without any backbone adaptation. The Phase A to Phase B transition produced an immediate gain of 6.21 percentage points in a single epoch (epoch 6: 96.34%), consistent with the hypothesis that the top backbone blocks encode high-level semantic features requiring

only minor domain-specific adjustment. By epoch 12, validation accuracy had stabilised at 99.44%, and the model converged to a best validation accuracy of **99.48%** at epoch 20.

## Test Set Performance

The model selected for evaluation corresponds to the checkpoint obtained at epoch 20, which achieved the highest *validation* accuracy of 99.48%. The *test set* accuracy — evaluated on the held-out 2,326-image partition — is **99.40%**, as reported in Table 2.10. Figure 2.8 presents the normalised confusion matrix and per-class  $F_1$  scores on the test set.

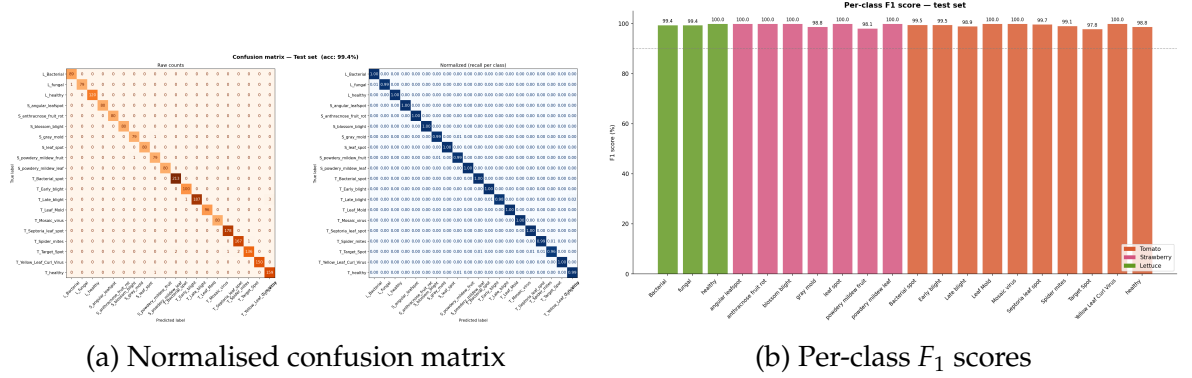


Figure 2.8: EfficientNet-B3 confusion matrix and per-class  $F_1$  scores on the test set.

The confusion matrix confirms that EfficientNet-B3 achieved near-perfect classification performance across all 20 disease categories. Most classes exhibit either perfect or near-perfect recognition, with only a small number of errors occurring between visually similar tomato diseases. The per-class  $F_1$  scores remain above 0.97 for all categories, indicating a highly balanced classifier with no significant performance degradation on minority classes.

Table 2.10 presents the complete per-class classification report for EfficientNet-B3.

Table 2.10: EfficientNet-B3 per-class precision, recall, and  $F_1$  on the test set (2,326 images, 20 classes).

| Plant                        | Class                  | Precision    | Recall       | $F_1$        | Support       |
|------------------------------|------------------------|--------------|--------------|--------------|---------------|
| Lettuce                      | Bacterial              | 0.989        | 1.000        | 0.994        | 89            |
|                              | Fungal                 | 1.000        | 0.988        | 0.994        | 80            |
|                              | Healthy                | 1.000        | 1.000        | 1.000        | 120           |
| Strawberry                   | Angular leafspot       | 1.000        | 1.000        | 1.000        | 80            |
|                              | Anthrachnose fruit rot | 1.000        | 1.000        | 1.000        | 80            |
|                              | Blossom blight         | 1.000        | 1.000        | 1.000        | 80            |
|                              | Gray mold              | 0.988        | 0.988        | 0.988        | 80            |
|                              | Leaf spot              | 1.000        | 1.000        | 1.000        | 80            |
|                              | Powdery mildew (fruit) | 0.975        | 0.988        | 0.981        | 80            |
|                              | Powdery mildew (leaf)  | 1.000        | 1.000        | 1.000        | 80            |
| Tomato                       | Bacterial spot         | 0.991        | 1.000        | 0.995        | 213           |
|                              | Early blight           | 0.990        | 1.000        | 0.995        | 100           |
|                              | Late blight            | 1.000        | 0.979        | 0.989        | 191           |
|                              | Leaf mold              | 1.000        | 1.000        | 1.000        | 96            |
|                              | Mosaic virus           | 1.000        | 1.000        | 1.000        | 80            |
|                              | Septoria leaf spot     | 0.994        | 1.000        | 0.997        | 178           |
|                              | Spider mites           | 0.988        | 0.994        | 0.991        | 168           |
|                              | Target spot            | 0.993        | 0.965        | 0.978        | 141           |
|                              | Yellow leaf curl virus | 1.000        | 1.000        | 1.000        | 150           |
|                              | Healthy                | 0.981        | 0.994        | 0.988        | 160           |
| <b>Macro average</b>         |                        | <b>0.994</b> | <b>0.995</b> | <b>0.995</b> | 2326          |
| <b>Weighted average</b>      |                        | <b>0.994</b> | <b>0.994</b> | <b>0.994</b> | 2326          |
| <b>Overall test accuracy</b> |                        |              |              |              | <b>99.40%</b> |

### 1.11 Three-Model Comparison

Table 2.11 consolidates the performance and efficiency metrics across all three models evaluated in this study.

Table 2.11: Three-model comparison — performance, efficiency, and training metrics.

| Metric                        | Custom CNN            | ResNet-50               | EfficientNet-B3  |
|-------------------------------|-----------------------|-------------------------|------------------|
| Test accuracy (%)             | <b>99.40</b>          | 99.44                   | <b>99.40</b>     |
| Macro $F_1$ (%)               | 99.3                  | 99.5                    | <b>99.5</b>      |
| Macro precision (%)           | 99.3                  | 99.4                    | 99.4             |
| Macro recall (%)              | 99.3                  | 99.5                    | 99.5             |
| Total parameters (M)          | <b>7.98</b>           | 25.6                    | 10.7             |
| Pretrained weights            | None                  | ImageNet                | ImageNet         |
| Input resolution              | $224 \times 224$      | $224 \times 224$        | $300 \times 300$ |
| Total epochs                  | 50                    | 20                      | <b>20</b>        |
| Avg. epoch time (s)           | $\approx 106$         | $\approx 116$           | $\approx 177$    |
| Total training time (min)     | $\approx 88$          | $\approx 39$            | $\approx 59$     |
| Epochs to reach 99% val. acc. | 37                    | 9                       | <b>10</b>        |
| Lowest class $F_1$ (%)        | 97.2<br>(L_Bacterial) | 98.6<br>(T_Target_Spot) | 97.8 (T_Target)  |
| Classes at $F_1 = 1.000$      | 9/20                  | 7/20                    | 9/20             |

All three models achieved strong performance on the 20-class heterogeneous corpus. EfficientNet-B3 and the Custom CNN both reached **99.40%** test accuracy, while ResNet-50 slightly surpassed both with **99.44%**. The absence of cross-domain degradation across studio and real-field images confirms that the preprocessing pipeline — domain-agnostic augmentation, colour normalisation, and stratified splitting — successfully bridged the acquisition-condition gap between the three source datasets.

The Custom CNN’s result of 99.40% significantly exceeds the 85–92% range reported in the literature [63], and is attributed to four design decisions: residual connections, Global Average Pooling, stable training dynamics via He initialisation and AdamW, and label smoothing ( $\varepsilon = 0.1$ ). However, Transfer learning significantly accelerated convergence: ResNet-50 reached 99% validation accuracy at epoch 9 ( $4.1\times$  faster than the CNN’s epoch 37), and EfficientNet-B3 at epoch 10 ( $3.7\times$  faster), confirming transfer learning as the preferred strategy in resource-constrained settings.

Per-class analysis identifies Tomato Target Spot and Lettuce Bacterial as the weakest classes across all models, both difficulties being data-level rather than architecture-level. Conversely, Strawberry Anthracnose Fruit Rot (87.9% synthetic images) achieved  $F_1 = 1.000$ , validating the augmentation pipeline design and the exclusion of CutMix and MixUp.

### Visual Test Samples and Model Justification

The Custom CNN was selected as the baseline architecture for three practical reasons. First, it was designed and trained *from scratch* on our specific dataset, without relying on external pretrained weights, which validates its generalisation capacity independently. Second, with only **7.98M parameters** — the smallest footprint among the three evaluated models — it is the most suitable candidate for future deployment on the embedded Raspberry Pi 5 of the VertiFarm platform. Third, its residual connections, Global Average Pooling, and label smoothing collectively produce stable training dynamics and strong per-class discrimination, as evidenced by the 99.40% test accuracy reported in Table 2.8.

The following images illustrate representative test samples drawn from the three plant categories, along with the label predicted by the CNN. These images are raw test-set samples

passed through the trained LeafDiseaseCNN model in the Kaggle Notebook environment; they are not screenshots of the deployed dashboard interface. The confidence values shown correspond to the Softmax output of the trained model on these held-out test images.

*Note on deployment:* the trained model weights (.pth) were exported and loaded into the VertiFarm platform running on the Raspberry Pi 5. Inference on the Pi uses the same PyTorch model in CPU evaluation mode. Conversion to a quantized or ONNX format to improve inference latency on the Pi 5 is identified as a direction for future work.

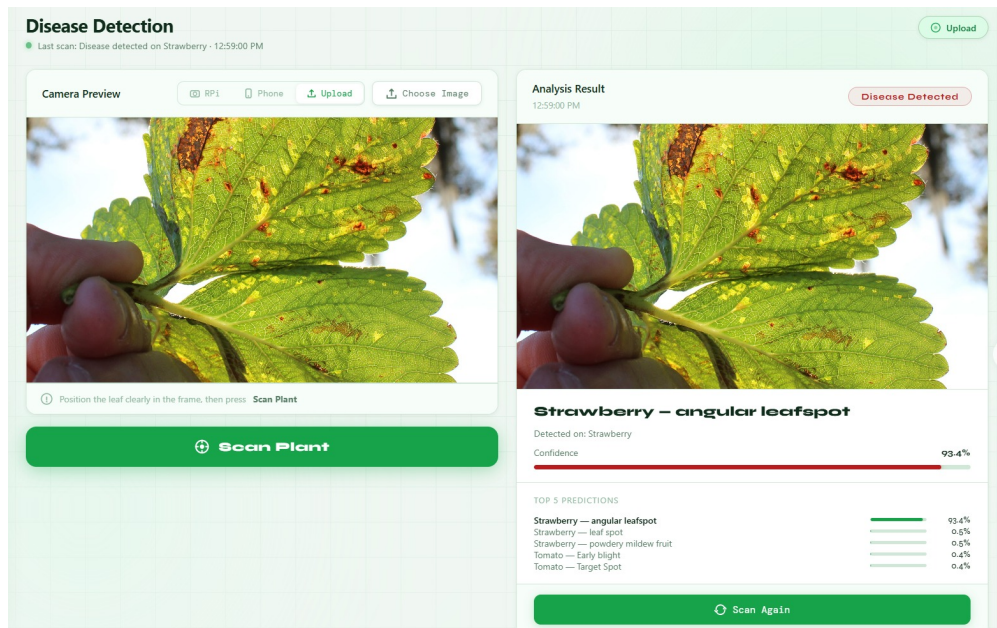


Figure 2.9: *Strawberry* — angular leafspot. **Predicted:** angular leafspot.

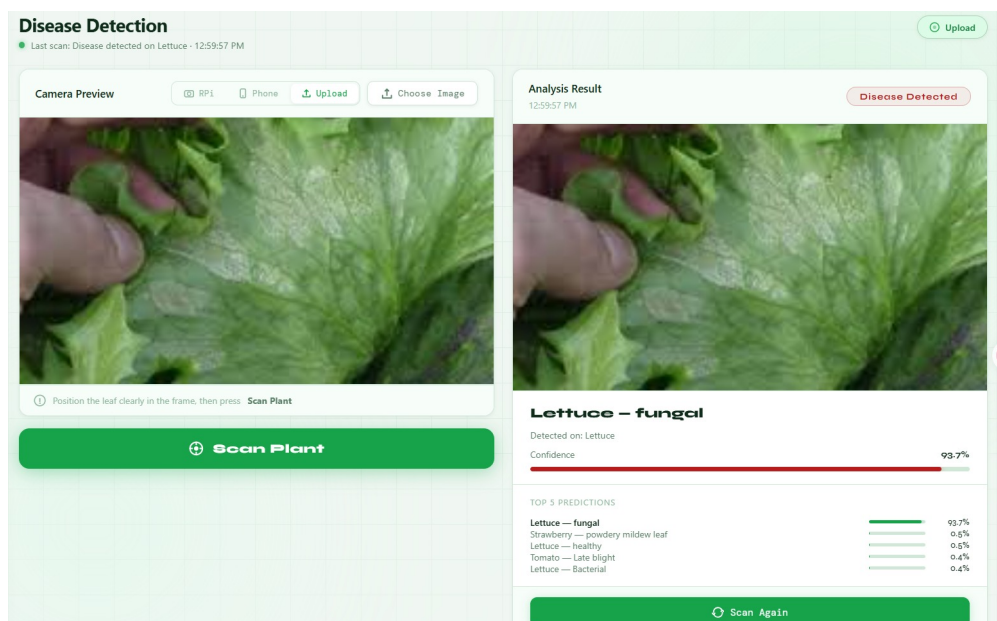


Figure 2.10: *Lettuce* — fungal. **Predicted:** fungal.

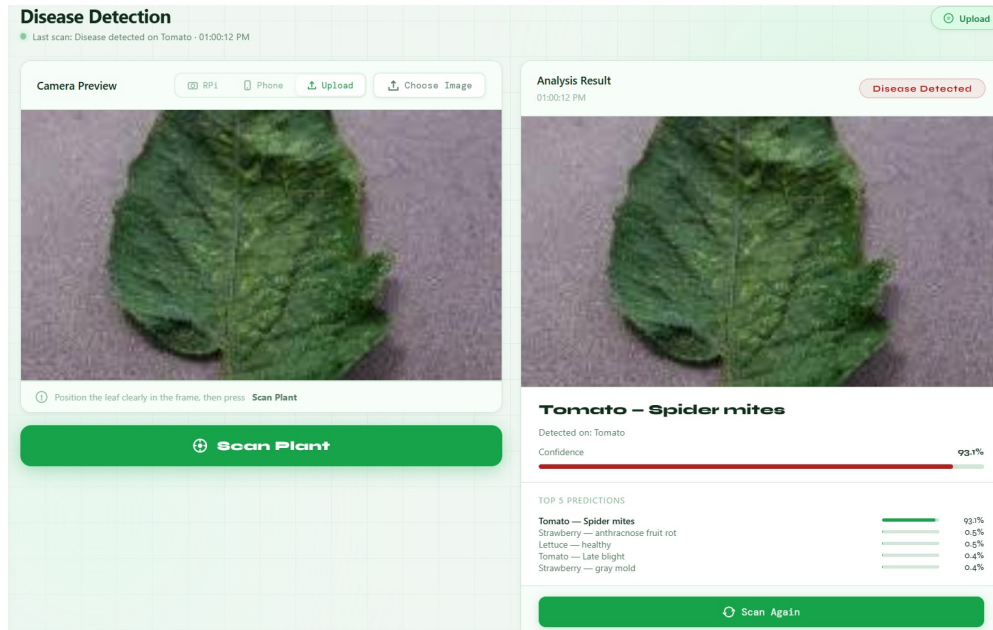


Figure 2.11: *Tomato* — spider mites. **Predicted:** spider mites.

## 2 Tomato Instance Segmentation Using YOLO11

To address the limitations of traditional detection methods, this section introduces a deep learning–based instance segmentation approach using the YOLO11 segmentation architecture for tomato detection and ripeness classification. This approach enables simultaneous object detection, classification, and pixel-level segmentation, allowing the system to accurately localize tomatoes within an image, determine their ripeness stage, and distinguish individual fruits even in crowded or complex agricultural scenes. Instance segmentation was selected instead of conventional object detection because tomatoes often overlap or appear clustered in real cultivation environments. By predicting segmentation masks rather than relying only on bounding boxes, the model provides more precise object boundaries and improves the separation of individual tomatoes, leading to more reliable ripeness assessment.

A heuristic yield estimation step is applied post-inference: detected tomatoes are counted per ripeness class and multiplied by a fixed average weight per class (green: 70 g, half-ripened: 90 g, fully ripened: 120 g), values derived from standard agronomic literature for tomato fruit at each ripeness stage. It is important to note that yield estimation is therefore a post-inference calculation, not a learned model output; the YOLO11 model itself performs instance segmentation and ripeness classification only.

The overall architecture of the proposed approach is illustrated in Figure 2.12. As shown in this figure, the input image is first processed by the backbone, which extracts hierarchical visual features such as color, texture, shape, and spatial information. These features are then transferred to the neck, where multi-scale feature fusion is performed to improve the detection of tomatoes with different sizes and positions. Finally, the detection head simultaneously predicts bounding boxes, ripeness class labels, objectness scores, and segmentation masks. Thus, the model is able to detect and localize each tomato, classify it as green, half-ripened, or fully ripened, and generate a pixel-level mask for each individual fruit. The YOLO11s-seg variant was selected as the optimal trade-off between accuracy and computational efficiency, comprising **10,083,449** parameters and operating at **33.1 GFLOPs** at an input resolution of  $640 \times 640$  pixels. Inference speed on the validation set averaged

20.6 ms per image on the Tesla T4 GPU, confirming suitability for near-real-time greenhouse monitoring.

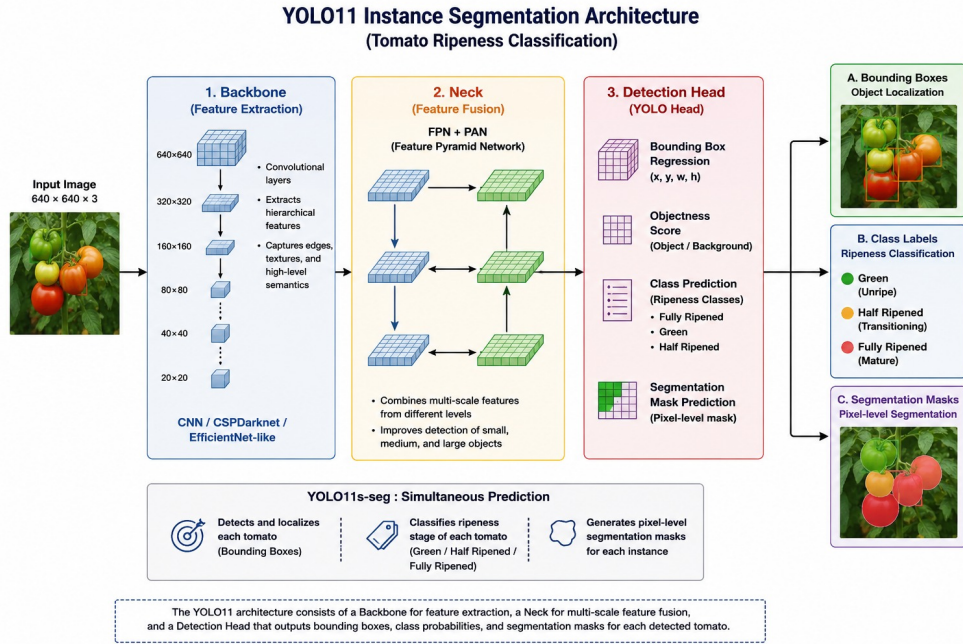


Figure 2.12: Yolo V11 Instance segmentation Architecture

## 2.1 Dataset Description

The dataset used in this study is the **Laboro Tomato Dataset**, obtained from Roboflow Universe. The dataset is designed for tomato detection and ripeness classification using instance segmentation annotations.

### Dataset Characteristics

- Total images: 2,174
- Training images: 2,001 (92.0%)
- Validation images: 86 (3.95%)
- Test images: 87 (4.0%)
- Annotation format: YOLO segmentation masks
- Image resolution: 640 × 640 pixels

Each tomato instance is annotated using polygonal masks, allowing precise segmentation of individual fruits.

The train/validation/test partition was adopted as provided by the Laboro Tomato dataset on Roboflow Universe [64] and was not modified, in order to preserve reproducibility with respect to other works using the same benchmark. The validation set of 86 images contains a total of 898 annotated instances (163 fully ripened, 568 green, 167 half-ripened), providing sufficient instance-level coverage for per-class metric evaluation despite the small image count.

## Classes

The dataset contains three tomato ripeness categories:

- **Fully Ripened** – mature red tomatoes ready for harvest
- **Green** – unripe tomatoes
- **Half Ripened** – tomatoes transitioning from green to red



(a) Fully Ripened



(b) Green



(c) Half Ripened

Figure 2.13: Examples of the three tomato ripeness categories from the Laboro Tomato Dataset.

This multi-task architecture allows the model to perform detection and segmentation simultaneously.

## 2.2 Training Configuration

The model was trained using the following configuration:

Table 2.12: YOLO11s-seg training configuration.

| Parameter              | Value                    |
|------------------------|--------------------------|
| <b>Model</b>           | YOLO11s-seg (pretrained) |
| <b>Image size</b>      | $640 \times 640$ px      |
| <b>Batch size</b>      | 8                        |
| <b>Epochs</b>          | 100                      |
| <b>Early stopping</b>  | Patience = 50 epochs     |
| <b>Optimizer</b>       | AdamW                    |
| <b>Learning rate</b>   | $1.429 \times 10^{-3}$   |
| <b>Training device</b> | NVIDIA Tesla T4 GPU      |

During training, the model optimizes several loss functions:

- **Box Loss:** measures bounding box localization error
- **Segmentation Loss:** measures accuracy of predicted masks
- **Classification Loss:** measures classification accuracy
- **DFL Loss:** improves bounding box localization precision

Lower loss values indicate better model performance.

## 2.3 Experimental Results

The final evaluation results of the best trained model are presented in Table 2.13.

Table 2.13: Overall Model Performance

| Metric        | Bounding Box | Segmentation Mask |
|---------------|--------------|-------------------|
| Precision     | 0.835        | 0.834             |
| Recall        | 0.848        | 0.847             |
| mAP@0.50      | 0.909        | 0.907             |
| mAP@0.50:0.95 | 0.809        | 0.766             |

### F1-Confidence Curve

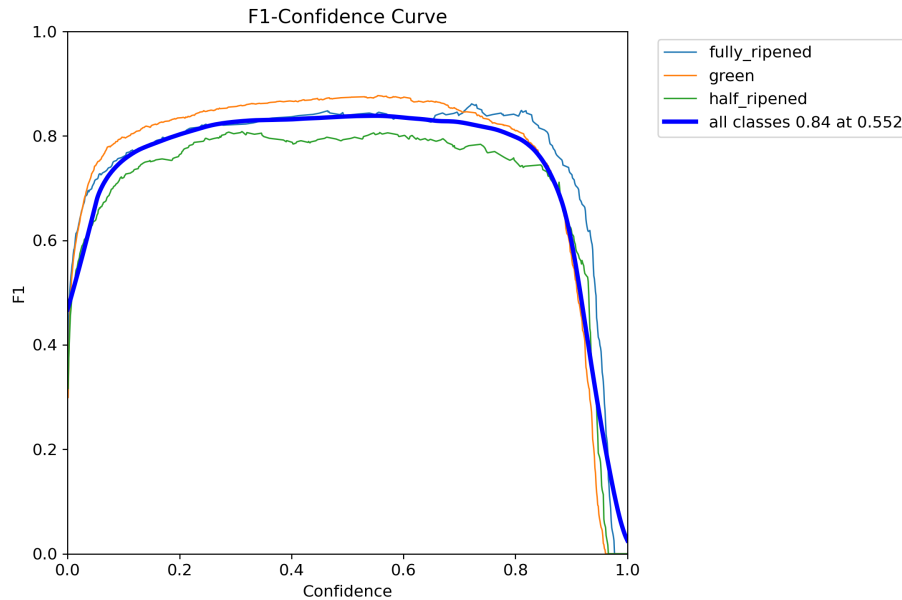


Figure 2.14: Bounding Box F1-Confidence Curve for YOLO11s-seg

Figure 2.14 shows the F1-Confidence curve, which measures the harmonic mean of precision and recall across all confidence thresholds. The model achieves a peak F1 score of **0.84** at an optimal confidence threshold of **0.552** across all classes, indicating a well-balanced detection performance.

The *green* class consistently achieves the highest F1 values throughout the confidence range, peaking near **0.88**, which reflects the model's strong ability to detect unripe tomatoes owing to their visually distinct coloration. The *fully\_ripened* class follows closely, maintaining a stable plateau above 0.84 across a wide confidence range (0.2–0.8), before sharply declining beyond 0.85. The *half\_ripened* class exhibits the lowest F1 curve, plateauing around 0.80, which is consistent with the inherent visual ambiguity of transitional-stage tomatoes that share color characteristics with both adjacent classes.

Notably, all three curves maintain a broad and stable plateau between confidence values of 0.2 and 0.7, demonstrating that the model is robust to the choice of threshold within this range. The sharp drop in F1 above a confidence of 0.85 is expected, as very few detections pass such a strict threshold.

## Recall-Confidence Curve

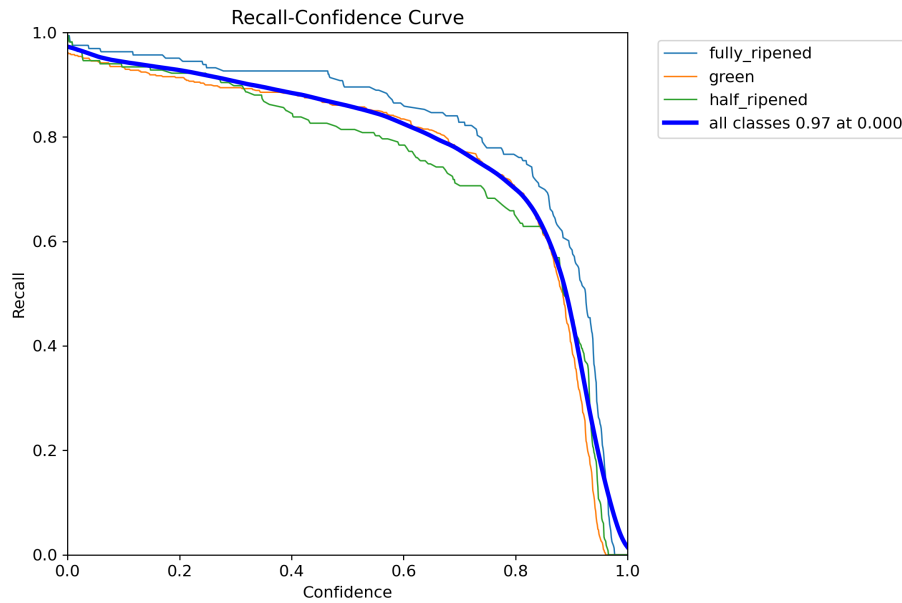


Figure 2.15: Bounding Box Recall-Confidence Curve for YOLO11s-seg

Figure 2.15 illustrates how recall evolves as the confidence threshold increases. At a threshold of **0.000**, the model attains a maximum recall of **0.97** across all classes, meaning that at low confidence thresholds the model successfully detects the vast majority of true tomato instances.

The *fully\_ripened* class achieves the highest recall overall, beginning near 0.97 and maintaining a notably slower rate of decay compared to the other classes. Its curve remains above 0.85 until approximately confidence 0.7, reflecting the model's high sensitivity toward mature red tomatoes. The *green* and *half\_ripened* classes exhibit steeper declines past confidence 0.5, with *green* dropping rapidly beyond 0.8 and reaching near-zero recall at 0.95. This suggests that the model assigns moderately high but not extreme confidence scores to green tomato detections.

The *half\_ripened* class shows the most irregular recall decay, indicating greater uncertainty in detection confidence for this class, which is attributable to its transitional appearance. Overall, the curve confirms that the model has a high sensitivity and would be suitable for applications prioritizing recall, such as automated harvesting where missing a tomato is more costly than a false detection.

## Precision-Recall Curve

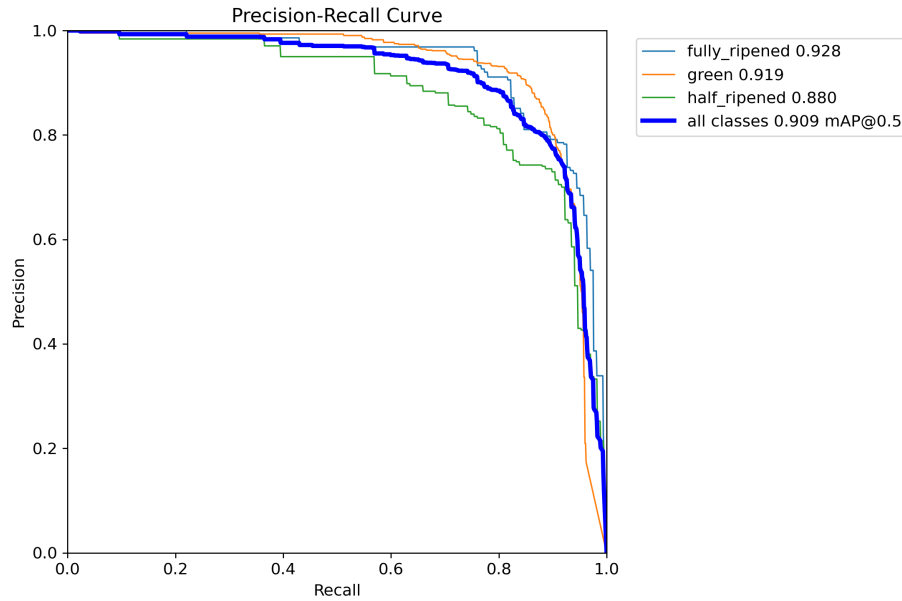


Figure 2.16: Bounding Box Precision-Recall Curve for YOLO11s-seg

Figure 2.16 presents the Precision-Recall (PR) curve, which characterizes the trade-off between precision and recall across detection thresholds. The overall mean Average Precision at IoU threshold 0.50 reaches  $\text{mAP@0.50} = 0.909$ , demonstrating strong detection performance across all three ripeness classes.

All three classes exhibit near-perfect precision (close to 1.0) at low recall values, indicating that when the model is most confident, its detections are almost exclusively correct. As recall increases, precision gradually decreases, with all curves maintaining high precision (above 0.9) for recall values up to approximately 0.8.

The *fully\_ripened* class achieves the largest area under the PR curve with an AP of **0.928**, confirming that mature red tomatoes are the most reliably detected class. This is likely due to their saturated red color, which provides a strong and unambiguous visual cue. The *green* class follows with an AP of **0.919**; its curve remains close to the *fully\_ripened* class throughout, with a slightly steeper precision drop at high recall values. The *half\_ripened* class achieves an AP of **0.880**, and its PR curve shows the earliest and most pronounced precision degradation, particularly beyond a recall of 0.6. This confirms that intermediate ripeness stages are the most difficult to detect accurately, as the model must distinguish tomatoes that simultaneously exhibit both green and red visual features.

## Confusion Matrix

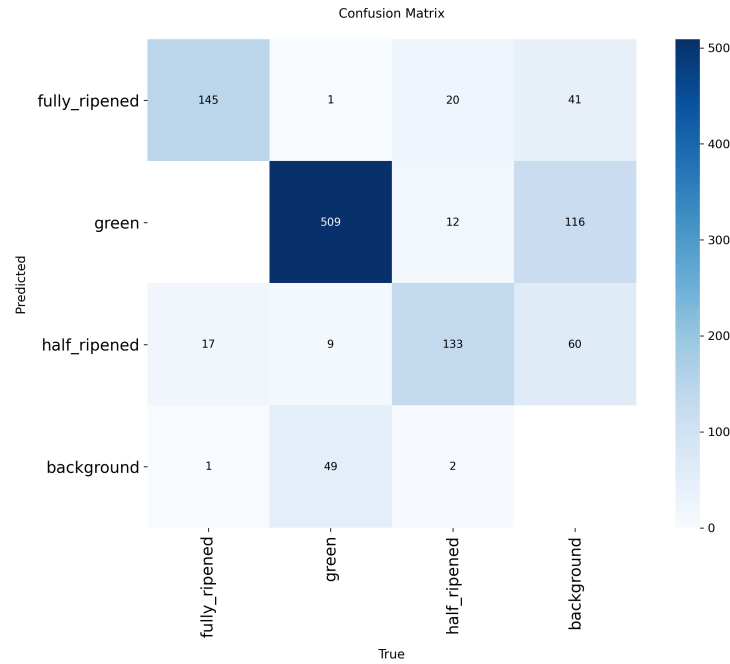


Figure 2.17: Confusion Matrix for YOLO11s-seg on the Validation Set

Figure 2.17 presents the absolute confusion matrix evaluated on the validation set. The diagonal elements represent correctly classified instances, while off-diagonal elements indicate misclassifications.

The validation set comprises **898** annotated tomato instances: 163 fully ripened, 568 green, and 167 half-ripened, evaluated across 86 validation images.

The *green* class achieves the strongest true positive count with **509** correct detections out of 568 true instances. Only 9 green instances were misclassified as *half\_ripened*, while 49 were missed entirely (predicted as background), reflecting occasional missed detections in cluttered regions. The model over-detected 116 background regions as green, which accounts for its lower precision relative to recall.

The *fully\_ripened* class correctly identifies **145** out of 163 true instances. Of the remaining 18 misclassified instances, 17 were predicted as *half\_ripened* and 1 was predicted as background. Additionally, the model incorrectly predicted *fully\_ripened* for 20 true *half\_ripened* instances and 41 true background regions, which explains the elevated false positive count for this class.

The *half\_ripened* class records **133** true positives out of 167 instances. Of the 34 missed instances, 20 were predicted as *fully\_ripened*, 12 as *green*, and 2 as background. The model also over-detects 60 true background regions as *half\_ripened* and 9 true green instances as *half\_ripened*, reflecting the visual ambiguity of transitional tomatoes whose colour overlaps with both adjacent ripeness stages.

Overall, the confusion matrix confirms that the model performs best on the *green* class (568 true instances, strong diagonal), and faces its greatest localisation challenge on *fully\_ripened* and *half\_ripened* classes, where background confusion is the dominant error mode rather than inter-class misclassification.

## 2.4 Class-wise Performance

Table 2.14 presents the segmentation performance for each ripeness class.

Table 2.14: Segmentation Performance per Class

| Class         | Precision | Recall | mAP@0.50 | mAP@0.50:0.95 |
|---------------|-----------|--------|----------|---------------|
| Fully ripened | 0.801     | 0.890  | 0.928    | 0.806         |
| Green         | 0.900     | 0.849  | 0.917    | 0.739         |
| Half ripened  | 0.803     | 0.803  | 0.876    | 0.754         |

## 2.5 Tomato Yield Estimation

Beyond ripeness classification, the YOLO11s-seg model can be extended to estimate the total yield weight of detected tomatoes within a single image. Since each detected tomato is assigned a ripeness class, an average weight can be associated with each class based on typical agronomic values. This allows a rough but practical estimation of the total harvestable weight directly from the model output.

### Weight Estimation Approach

The estimation is based on assigning a fixed average weight to each ripeness category:

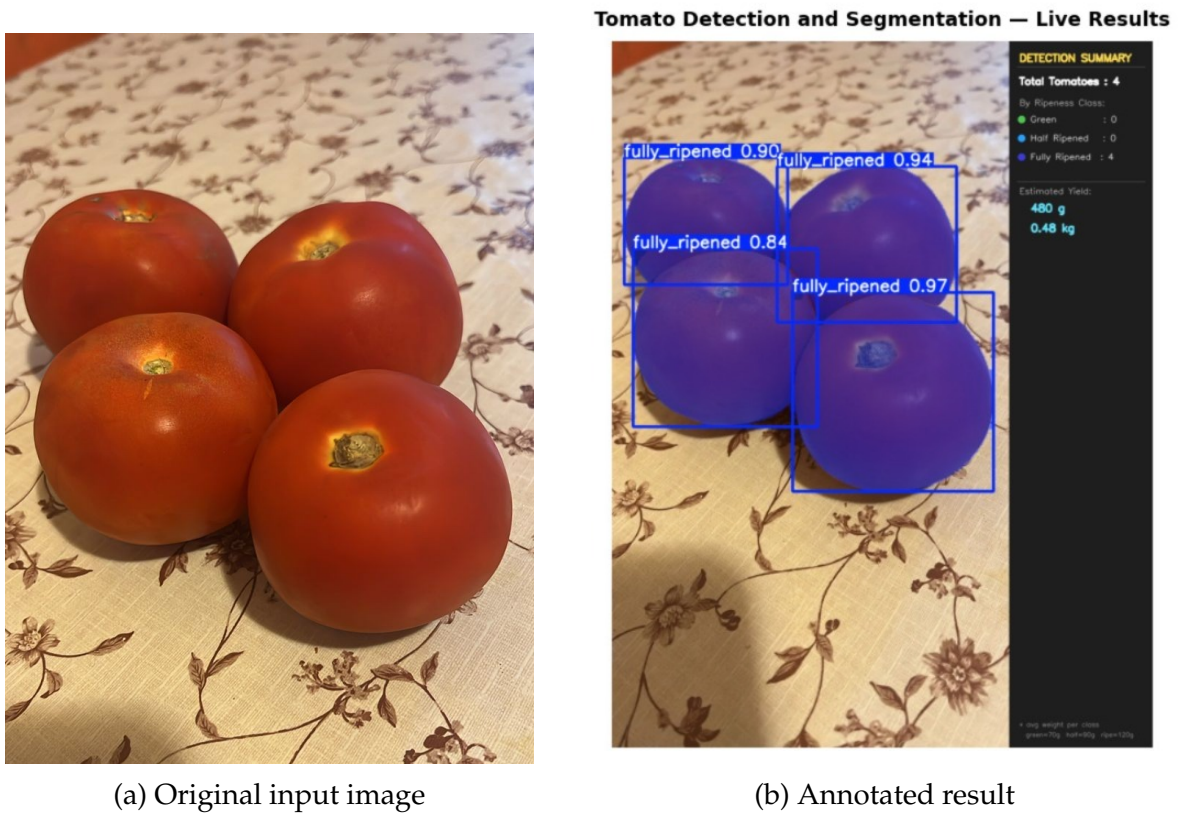
- **Green** (unripe): 70 g per tomato
- **Half Ripened** (transitioning): 90 g per tomato
- **Fully Ripened** (mature): 120 g per tomato

After inference, the total estimated weight is computed by summing the average weights of all detected instances across all classes, as shown in Equation 2.1:

$$W_{\text{total}} = \sum_{i=1}^N w_{c_i} \quad (2.1)$$

where  $N$  is the total number of detected tomatoes and  $w_{c_i}$  is the average weight associated with the ripeness class  $c_i$  of the  $i$ -th detection.

Figure 2.18 illustrates the estimation pipeline applied to a real greenhouse image. The original image (left) shows a cluster of tomatoes at various ripeness stages. After running inference with the trained YOLO11s-seg model, the annotated image (right) displays the detected instances with their predicted ripeness class labels, confidence scores, and bounding boxes. A summary panel overlaid on the result reports the per-class counts and the total estimated yield.



(a) Original input image

(b) Annotated result

Figure 2.18: Tomato yield estimation: (left) raw input image; (right) YOLO11s-seg

For the example shown in Figure 2.18, the model detected a total of **4 tomatoes**, all classified as fully ripened. Applying the average weight per class yields a total estimated weight of:

$$W_{\text{total}} = 4 \times 120 = 480 \text{ g} \approx 0.48 \text{ kg} \quad (2.2)$$

The results demonstrate that the YOLO11s-seg model performs effectively for tomato instance segmentation and ripeness classification. The model achieved a segmentation  $\text{mAP}@0.50$  of 90.7%, indicating strong accuracy in detecting and segmenting tomato instances. The stricter metric  $\text{mAP}@0.50:0.95$  reached 76.6%, which shows good performance even under stricter overlap requirements. Among the classes, fully ripened tomatoes achieved the highest segmentation accuracy, with a  $\text{mAP}@0.50:0.95$  score of 0.812. This suggests that the model can more easily recognize fully mature tomatoes due to their distinctive color and texture. Green tomatoes achieved the highest precision (0.900), indicating fewer false detections for this class. The half-ripened class showed slightly lower performance, likely because these tomatoes share visual features with both green and fully ripened tomatoes, making them harder to distinguish. Overall, the model demonstrates strong capability in detecting and segmenting tomatoes in agricultural scenes. The segmentation approach also improves object separation when tomatoes are clustered together, which is common in real greenhouse environments.

## 2.6 Conclusion

This chapter presented the design and implementation of advanced deep learning models for agricultural monitoring, combining both plant disease classification and tomato detection with ripeness analysis. Three deep learning architectures were evaluated for plant leaf disease

classification across 20 disease classes spanning tomato, strawberry, and lettuce. All three models achieved strong performance, with test accuracies between 99.40% and 99.44%, demonstrating the robustness and reliability of the proposed pipeline across heterogeneous acquisition conditions.

In parallel, the YOLO11 segmentation model proved to be a highly effective solution for automated tomato detection and ripeness classification. By integrating object detection with instance segmentation, the model enables precise localization, classification, and segmentation of tomatoes within complex environments.

The results obtained throughout this chapter highlight the effectiveness of combining classification and segmentation approaches to address key challenges in precision agriculture. This work demonstrates a strong capability to develop intelligent, data-driven systems for plant health monitoring and crop management. With further optimization and real-world deployment, the proposed solutions have significant potential to improve agricultural productivity, support automated harvesting, and contribute to the development of sustainable and smart farming systems.

## INTELLIGENT HYDROPONIC SYSTEM: FUZZY LOGIC CONTROL, DIGITAL TWIN MODELLING, AND CONVERSATIONAL AI

### Introduction

This chapter addresses the two remaining intelligent components that complete the platform: a Digital Twin integration layer and a domain-constrained conversational assistant, HydroBot.

The first part introduces the concept of Digital Twins and examines their application in smart agriculture, before presenting the specific three-layer architecture adopted for VertiFarm. It details how real-time sensor data is synchronised with a virtual representation of the physical system, how behavioural modelling is achieved through the Mamdani fuzzy logic controller, and how AI-based disease detection is embedded within the twin. The dashboard is then discussed as the operator-facing window onto this digital replica. A comparative review of related works concludes this part and positions VertiFarm within the current state of the art.

The second part presents HydroBot, a conversational assistant designed to support operators in managing the hydroponic system. Built around a locally hosted large language model served via Ollama, HydroBot integrates a domain-filtering mechanism and carefully engineered system prompts to constrain its responses strictly to hydroponics-related queries. Its architecture, safety mechanisms, inference pipeline, and user interface are described in detail, highlighting the design choices that make it suitable for offline deployment on the Raspberry Pi 5.

Together, these two components extend VertiFarm beyond simple monitoring and control, towards a fully integrated intelligent platform capable of self-representation, autonomous regulation, and natural-language interaction.

## 1 Digital Twin Integration in the VertiFarm Platform

### 1.1 Concept and Definition of Digital Twins

The concept of the Digital Twin was first formally introduced by Grieves [65] in the context of product lifecycle management and was later adopted by NASA for spacecraft simulation. Today it has become a cornerstone technology in Industry 4.0, cyber-physical systems, and smart agriculture.

A *Digital Twin* (DT) is a dynamic, continuously updated virtual replica of a physical system that mirrors its real-world counterpart in real time through bidirectional data exchange. It integrates sensor data, physical models, and intelligent algorithms to simulate, monitor, predict, and autonomously control the behaviour of its physical twin [66].

Three fundamental components constitute a Digital Twin architecture [65,66]:

1. **Physical entity** — the real-world system instrumented with sensors and actuators.
2. **Virtual entity** — the computational model that replicates the physical system’s state, behaviour, and history.
3. **Data connection** — the bidirectional communication channel through which the physical entity updates the virtual model and the virtual model sends control decisions back to the physical entity.

Figure 3.1 illustrates the general Digital Twin architecture.

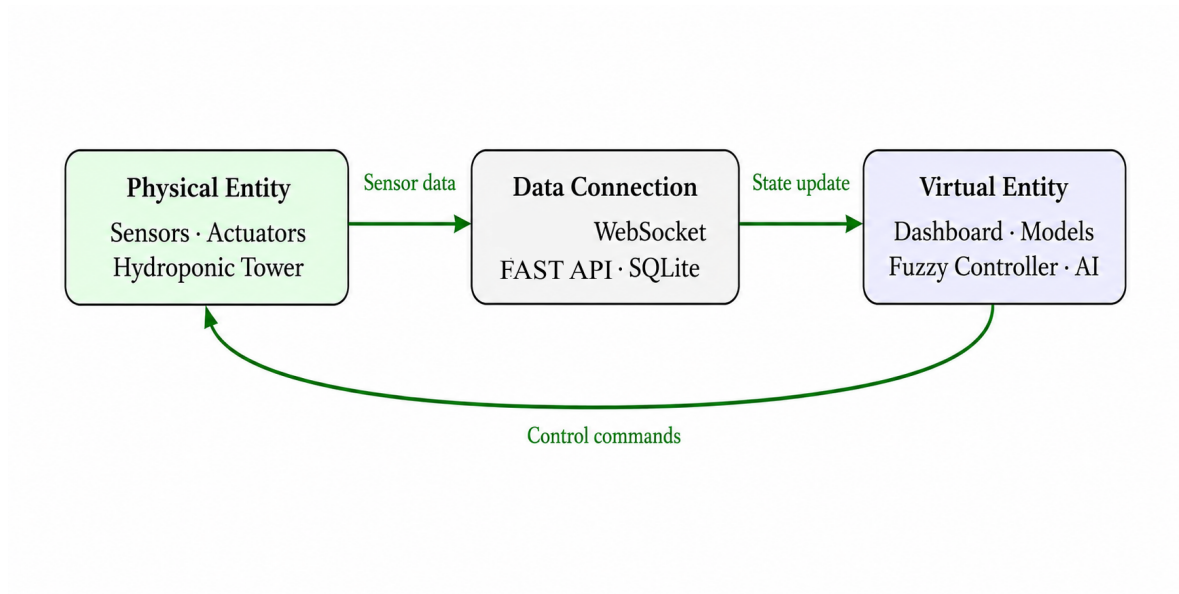


Figure 3.1: General Digital Twin architecture

## 1.2 Digital Twins in Smart Agriculture

The application of Digital Twins to precision agriculture has gained significant attention in recent years. Researchers have demonstrated that DT frameworks can improve resource efficiency in controlled-environment agriculture: Alinejad et al. [67] reported a 28 % reduction in water consumption by coupling sensor data with an evapotranspiration model, while broader reviews confirm improvements in yield prediction accuracy and proactive disease management [68]. In hydroponic systems specifically, the controlled environment and continuous sensor instrumentation make them ideal candidates for a full DT implementation, since every environmental parameter can be monitored and actuated without the variability present in open-field agriculture [69].

## 1.3 The VertiFarm Digital Twin Architecture

The VertiFarm platform constitutes a purpose-built Digital Twin for a hydroponic tower growing system. It realises the three foundational DT components described in Section 1.1 as follows.

## Physical Entity

The physical twin consists of a Raspberry Pi 5 single-board computer connected to the hydroponic tower via the I<sup>2</sup>C bus. Table 3.1 summarises the instrumentation layer.

Table 3.1: Physical entity instrumentation of the VertiFarm Digital Twin.

| Component      | Module               |    | Interface               | Function                             |
|----------------|----------------------|----|-------------------------|--------------------------------------|
| pH probe       | PH-4502C<br>ADS1115  | +  | I <sup>2</sup> C (0x48) | Measures nutrient solution acidity   |
| EC/TDS probe   | TDS V1.0<br>ADS1115  | +  | I <sup>2</sup> C (0x48) | Measures nutrient concentration      |
| Environmental  | BME280               |    | I <sup>2</sup> C (0x76) | Air temperature, humidity, pressure  |
| Camera         | RPi Camera<br>Module |    | CSI                     | Captures leaf images for AI analysis |
| pH-Up valve    | Solenoid<br>1/2"     | NC | GPIO                    | Raises solution pH (83 mL/s)         |
| pH-Down valve  | Solenoid<br>1/2"     | NC | GPIO                    | Lowers solution pH (83 mL/s)         |
| Nutrient valve | Solenoid<br>1/2"     | NC | GPIO                    | Dispenses nutrient concentrate       |

## Virtual Entity

The virtual twin is the VertiFarm software platform running on the same Raspberry Pi. It maintains a continuously updated computational representation of the physical system through four interconnected layers:

- **State layer** — the in-memory LATEST dictionary in `services.py` holds the most recent sensor snapshot at all times, updated every 2 seconds by the background sensor loop. This constitutes the *live state* of the virtual twin, mirroring the current physical condition of the reservoir with minimal latency.
- **History layer** — the SQLite database (`VertiFarm.db`) persists time-series measurements every 10 seconds into the `measurements` table. This constitutes the *historical model* of the system, enabling retrospective analysis, trend identification, and feeding data to the AI inference pipeline.
- **Behavioural model layer** — the fuzzy logic controller in `fuzzy.py` embeds expert agronomic knowledge as a Mamdani inference system. Given the current state (pH, EC) and the selected crop profile, it computes the corrective dosing action the physical system should execute. This constitutes the *predictive model* of the twin.
- **Intelligent inference layer** — the LeafDiseaseCNN convolutional neural network in `ai.py` analyses images captured by the physical camera and classifies plant health across 20 disease categories. This constitutes the *anomaly detection model* of the twin.

## Data Connection

Bidirectional data exchange between the physical and virtual entities is implemented through two complementary mechanisms:

- **Physical → Virtual (upward flow)** — the `sensor_loop()` coroutine in `services.py` reads all sensors via the I<sup>2</sup>C bus every 2 seconds and updates the virtual state immediately. A WebSocket connection (`/ws/live`) propagates state changes to the browser-based dashboard within milliseconds of each hardware read.
- **Virtual → Physical (downward flow)** — the `control_loop()` coroutine reads the virtual state, evaluates the fuzzy inference model, and issues commands to the physical solenoid valves by calling `hardware.set_valve()` via Raspberry Pi GPIO. In manual mode, commands are issued directly by the operator through the dashboard interface.

## 1.4 Digital Twin Functional Layers

Figure 3.2 presents the complete layered architecture of the VertiFarm Digital Twin, mapping each software component to its corresponding DT functional role.

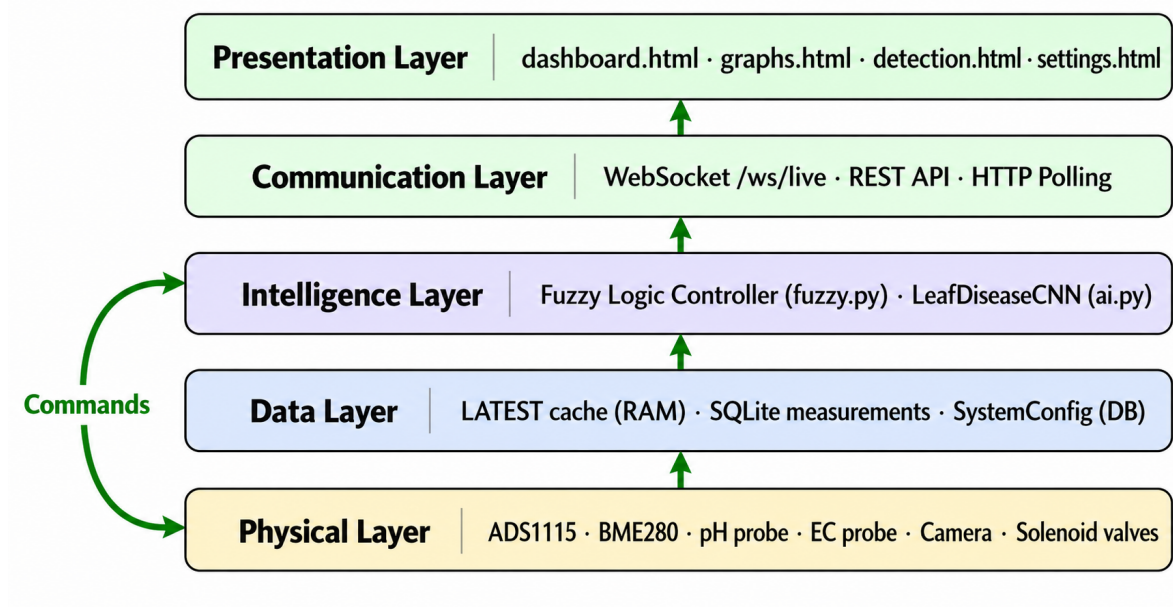


Figure 3.2: Layered architecture of the VertiFarm Digital Twin.

## 1.5 Real-Time State Synchronisation

A defining characteristic of a Digital Twin is the continuous synchronisation of the virtual model with the physical state. In VertiFarm, this is achieved through a two-speed design that balances responsiveness with hardware durability.

The sensor loop operates at a **2-second read interval**, updating the in-memory LATEST state cache on every iteration. This ensures that the virtual twin's state representation lags behind the physical system by at most 2 seconds, satisfying the near-real-time synchronisation requirement. The WebSocket endpoint propagates each state update to connected dashboard clients within milliseconds by awaiting an `asyncio.Event` signal:

```
await services.LATEST_EVENT.wait()    # suspend until new data
```

```
payload = build_dashboard_payload(db)
await ws.send_json(payload) # push to browser
```

The database write interval is decoupled from the read interval, operating at **10 second** intervals, which reduces SD card write cycles from 43 200 to 8 640 per day while preserving the live state in RAM for immediate access.

## 1.6 Behavioural Modelling — Fuzzy Logic Controller

The fuzzy logic controller constitutes the *behavioural model* of the Digital Twin — the component that encodes how the physical system *should* respond to deviations from the desired state. Unlike a simple threshold-based controller, the fuzzy model captures the continuous, graduated nature of chemical dosing.

Although the current implementation does not include a physics-based plant growth simulator or long-horizon forecasting model, it provides a form of behavioural prediction through fuzzy inference. Given the current state of the hydroponic system, the controller estimates the corrective actions required to drive the physical environment toward its desired operating conditions. In this sense, the virtual model does not merely represent the current state of the system but also computes its expected response to changing conditions, contributing to the predictive capabilities of the Digital Twin.

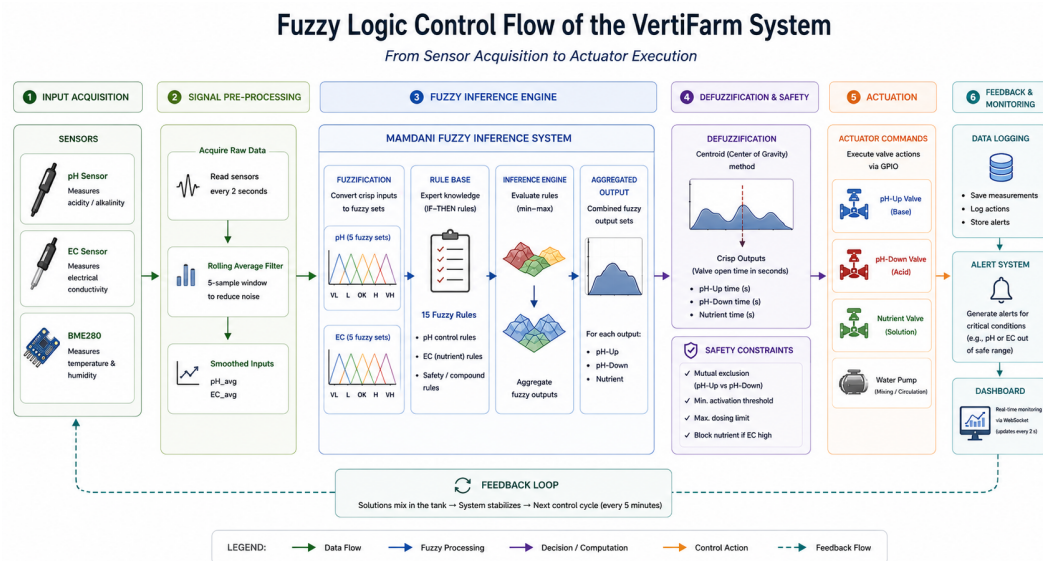


Figure 3.3: Fuzzy logic control flow of the VertiFarm system from sensor acquisition to actuator execution.

To ensure adaptive and stable regulation of pH and nutrient concentration in the hydroponic system, a Mamdani fuzzy logic controller was implemented. This approach is particularly suitable for hydroponic environments due to the nonlinear behavior of water chemistry and the presence of noisy sensor measurements.

The controller translates agronomic expert knowledge into a rule-based decision system that computes corrective actions for pH-Up, pH-Down, and nutrient dosing without requiring an explicit mathematical model of the system.

A custom fuzzy inference engine was implemented in pure Python to ensure lightweight execution on the Raspberry Pi and full control over the inference process.

## Control Loop Overview

The system operates in a periodic control loop executed every 5 minutes. Each cycle performs the following steps:

- Acquisition of pH and EC sensor readings.
- Application of a rolling average filter to reduce measurement noise.
- Evaluation of fuzzy rules to compute control actions.
- Execution of actuator commands through GPIO-controlled valves.
- Logging of all actions and generation of alerts for abnormal conditions.
- Stabilization delay to ensure proper mixing of injected solutions.

**Control interval rationale.** The 5-minute control interval was chosen based on two constraints specific to solenoid valve dosing in hydroponic reservoirs. First, chemical solutions injected into the reservoir require a mixing and diffusion period before a stable, representative pH or EC measurement can be obtained; a mixing delay shorter than 3–4 minutes risks triggering a second dosing cycle before the first injection has equilibrated, causing overshoot and oscillation around the setpoint. Second, frequent solenoid actuation accelerates mechanical wear on the valve seats. The 2-second sensor read interval serves a different purpose: it provides a high-resolution live state for the dashboard display and generates accurate rolling-average inputs for the fuzzy controller at each 5-minute evaluation cycle. In future work, a tiered control architecture could apply rapid intervention only when sensor readings cross critical safety thresholds, while maintaining the 5-minute cycle for routine regulation.

In autonomous operation, the fuzzy controller continuously evaluates sensor measurements and determines the required dosing actions without operator intervention, as shown in Figure 3.4.

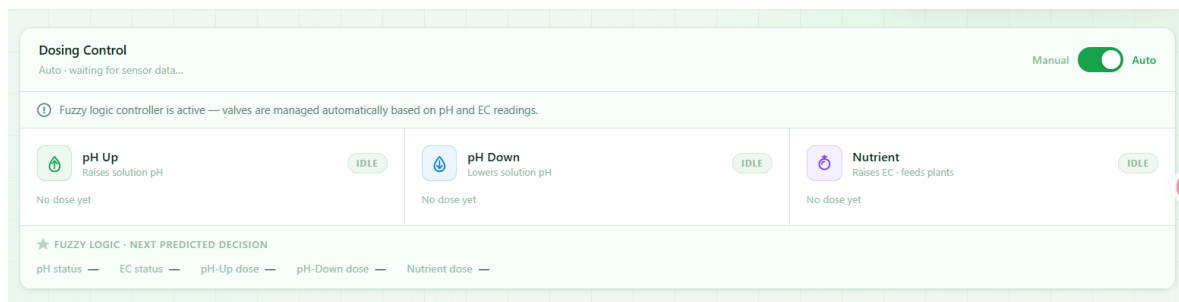


Figure 3.4: Automatic dosing mode using the fuzzy logic controller.

For maintenance, testing, or exceptional situations, the operator may switch to manual mode and actuate individual valves directly through the dashboard interface, as illustrated in Figure 3.5.

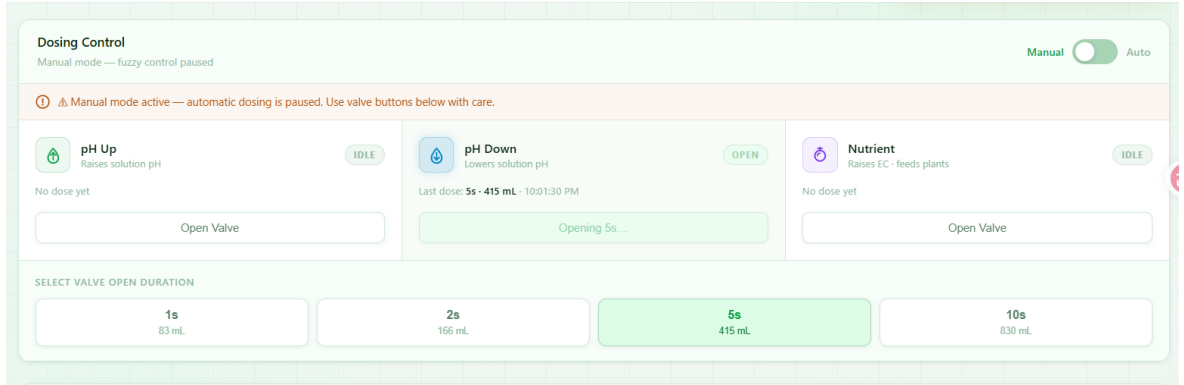


Figure 3.5: Manual dosing mode allowing direct operator control of valves.

## Membership Functions

The fuzzy inference system maps each crisp sensor reading to five linguistic variables using a combination of three membership function types implemented in pure Python (no external libraries):

- **Triangular** (`_trimf`): defined by parameters  $(a, b, c)$  where  $a$  is the left foot ( $\mu = 0$ ),  $b$  is the peak ( $\mu = 1$ ), and  $c$  is the right foot ( $\mu = 0$ ):

$$\mu_{\text{tri}}(x; a, b, c) = \begin{cases} 0 & x \leq a \text{ or } x \geq c \\ \frac{x-a}{b-a} & a < x \leq b \\ \frac{c-x}{c-b} & b < x < c \end{cases} \quad (3.1)$$

- **Z-shaped** (`_zmf`): used for the *very low* set — membership equals 1 below parameter  $a$  and decreases linearly to 0 at parameter  $b$ :

$$\mu_{\text{zmf}}(x; a, b) = \begin{cases} 1 & x \leq a \\ 1 - \frac{x-a}{b-a} & a < x < b \\ 0 & x \geq b \end{cases} \quad (3.2)$$

- **S-shaped** (`_smf`): used for the *very high* set — membership equals 0 below parameter  $a$  and increases linearly to 1 at parameter  $b$ :

$$\mu_{\text{smf}}(x; a, b) = \begin{cases} 0 & x \leq a \\ \frac{x-a}{b-a} & a < x < b \\ 1 & x \geq b \end{cases} \quad (3.3)$$

Each input variable (pH and EC) is partitioned into five linguistic fuzzy sets:

$$\{\text{very low}, \text{low}, \text{ok}, \text{high}, \text{very high}\}$$

The membership function boundaries are derived **dynamically** from the active `CropProfile` dataclass stored in the `SystemConfig` database table, and are automatically reconfigured when the operator switches crop profiles via the Settings page. Tables 3.2 and 3.3 list the exact parameters for all three supported crop profiles, as defined in `fuzzy.py`.

Table 3.2: pH input membership function parameters for all three crop profiles.  $\text{zmf}(a, b)$  = Z-shaped function (full membership below  $a$ , zero at  $b$ );  $\text{trimf}(a, b, c)$  = triangular (peak at  $b$ );  $\text{smf}(a, b)$  = S-shaped (zero at  $a$ , full membership above  $b$ ). All values are pH units.

| Fuzzy Set                                 | Function                | Lettuce         | Tomato          | Strawberry      |
|-------------------------------------------|-------------------------|-----------------|-----------------|-----------------|
| <i>very low</i>                           | $\text{zmf}(a, b)$      | (5.0, 5.8)      | (5.0, 5.8)      | (5.2, 5.8)      |
| <i>low</i>                                | $\text{trimf}(a, b, c)$ | (5.0, 5.8, 6.2) | (5.0, 5.8, 6.0) | (5.2, 5.8, 6.0) |
| <i>ok</i>                                 | $\text{trimf}(a, b, c)$ | (5.8, 6.2, 6.5) | (5.8, 6.0, 6.3) | (5.8, 6.0, 6.2) |
| <i>high</i>                               | $\text{trimf}(a, b, c)$ | (6.2, 6.5, 7.2) | (6.0, 6.3, 7.0) | (6.0, 6.2, 6.8) |
| <i>very high</i>                          | $\text{smf}(a, b)$      | (6.5, 7.2)      | (6.3, 7.0)      | (6.2, 6.8)      |
| <b>Optimal target (peak of <i>ok</i>)</b> |                         | <b>6.2</b>      | <b>6.0</b>      | <b>6.0</b>      |

Table 3.3: EC input membership function parameters for all three crop profiles. Function types are identical to Table 3.2. All values are in  $\text{mS}\cdot\text{cm}^{-1}$ .

| Fuzzy Set                                 | Function                | Lettuce         | Tomato          | Strawberry      |
|-------------------------------------------|-------------------------|-----------------|-----------------|-----------------|
| <i>very low</i>                           | $\text{zmf}(a, b)$      | (0.6, 1.2)      | (1.0, 2.0)      | (0.8, 1.4)      |
| <i>low</i>                                | $\text{trimf}(a, b, c)$ | (0.6, 1.2, 1.6) | (1.0, 2.0, 2.8) | (0.8, 1.4, 1.8) |
| <i>ok</i>                                 | $\text{trimf}(a, b, c)$ | (1.2, 1.6, 2.0) | (2.0, 2.8, 3.5) | (1.4, 1.8, 2.0) |
| <i>high</i>                               | $\text{trimf}(a, b, c)$ | (1.6, 2.0, 2.8) | (2.8, 3.5, 4.5) | (1.8, 2.0, 2.5) |
| <i>very high</i>                          | $\text{smf}(a, b)$      | (2.0, 2.8)      | (3.5, 4.5)      | (2.0, 2.5)      |
| <b>Optimal target (peak of <i>ok</i>)</b> |                         | <b>1.6</b>      | <b>2.8</b>      | <b>1.8</b>      |

**Scientific basis.** The crop profile boundaries are grounded in established hydroponic agronomy references: Resh [70] for Lettuce and Tomato optimal ranges, Hoagland & Arnon [71] and FAO [72] for nutrient solution guidelines, These values are stored in the SystemConfig database table and can be updated without modifying the inference engine code.

## Output Membership Functions

The output variable (valve open time in seconds) is partitioned into six linguistic actuation levels: *none*, *tiny*, *small*, *medium*, *large*, and *max*. The central levels are represented using triangular membership functions, while the boundary levels (*none* and *max*) are implemented using left- and right-shoulder functions to ensure smooth saturation at the limits of the control range, as defined in `_output_mf()` and evaluated over the discrete universe  $\mathcal{U} = \{0.0, 0.5, 1.0, 2.0, 3.0, 5.0, 7.0, 10.0\}$  seconds.

The output membership functions were calibrated experimentally based on the measured flow rate of the solenoid valves (83 mL/s). The selected actuation levels provide progressively stronger corrective actions while maintaining stable system behaviour and preventing abrupt chemical changes within the nutrient reservoir.

Table 3.4: Output membership functions for valve dosing time (seconds). All six levels use triangular  $\text{trimf}(a, b, c)$  functions except *none* (left-edge, implemented as  $\text{zmf}$ ) and *max* (right-edge, implemented as  $\text{smf}$ ). Volume computed at flow rate = 83 mL/s.

| Level         | Parameters ( $a, b, c$ ) | Peak time (s) | Volume at peak (mL) | Safety role                 |
|---------------|--------------------------|---------------|---------------------|-----------------------------|
| <i>none</i>   | (0.0, 0.0, 0.5)          | 0.0 s         | 0 mL                | No action                   |
| <i>tiny</i>   | (0.0, 0.5, 1.0)          | 0.5 s         | 41.5 mL             | Below MIN_DOSE — suppressed |
| <i>small</i>  | (0.5, 1.0, 2.0)          | 1.0 s         | 83 mL               | Minor correction            |
| <i>medium</i> | (1.0, 2.0, 4.0)          | 2.0 s         | 166 mL              | Moderate correction         |
| <i>large</i>  | (2.0, 5.0, 7.0)          | 5.0 s         | 415 mL              | Strong correction           |
| <i>max</i>    | (5.0, 10.0, 10.0)        | 10.0 s        | 830 mL              | Maximum dose (MAX_DOSE cap) |

**Safety constants.** Two hard limits are enforced by the `defuzzify()` function regardless of fuzzy output:

- $\text{MIN\_DOSE\_S} = 0.5 \text{ s}$  (= 41.5 mL): crisp outputs below this threshold are set to zero, suppressing negligible doses that would cause actuator wear without meaningful chemical effect. This is why the *tiny* output level effectively acts as a dead-band.
- $\text{MAX\_DOSE\_S} = 10.0 \text{ s}$  (= 830 mL): crisp outputs are capped at this value to prevent over-correction in a single cycle regardless of the degree of deviation.

## Rule-Based Control

The system uses a set of expert-defined fuzzy rules implemented in the `evaluate_rules()` function of `fuzzy.py`. The rules treat pH correction and EC (nutrient) correction as two independent axes, each evaluated against its own set of five linguistic states. A single compound rule handles the case where both inputs are simultaneously critical. The complete rule base is presented in Table 3.5.

Table 3.5: Fuzzy rule base implemented in `evaluate_rules()` — logical conditions, activated output levels, and indicative centroid dose values (Lettuce profile, 83 mL/s flow rate).

| Rule                                                              | Condition (Input)                                  | Output levels activated              | Indicative time (s)                                            | Indicative volume (mL) |
|-------------------------------------------------------------------|----------------------------------------------------|--------------------------------------|----------------------------------------------------------------|------------------------|
| <i>pH-Up rules — raise pH when solution is too acidic</i>         |                                                    |                                      |                                                                |                        |
| R1                                                                | IF pH is <i>ok</i>                                 | → pH-Up: NONE                        | 0.00                                                           | 0                      |
| R2                                                                | IF pH is <i>high</i> or <i>very high</i>           | → pH-Up: NONE (wrong direction)      | 0.00                                                           | 0                      |
| R3                                                                | IF pH is <i>low</i>                                | → pH-Up: SMALL                       | 1.00                                                           | 83                     |
| R4                                                                | IF pH is <i>very low</i>                           | → pH-Up: LARGE (×0.7 strength)       | —                                                              | —                      |
| R5                                                                | IF pH is <i>very low</i>                           | → pH-Up: MAX (×0.5 strength)         | 5.14*                                                          | 427*                   |
| <i>pH-Down rules — lower pH when solution is too alkaline</i>     |                                                    |                                      |                                                                |                        |
| R6                                                                | IF pH is <i>ok</i>                                 | → pH-Down: NONE                      | 0.00                                                           | 0                      |
| R7                                                                | IF pH is <i>low</i> or <i>very low</i>             | → pH-Down: NONE (wrong direction)    | 0.00                                                           | 0                      |
| R8                                                                | IF pH is <i>high</i>                               | → pH-Down: SMALL                     | 2.27                                                           | 188                    |
| R9                                                                | IF pH is <i>very high</i>                          | → pH-Down: MEDIUM (×0.7 strength)    | —                                                              | —                      |
| R10                                                               | IF pH is <i>very high</i>                          | → pH-Down: LARGE (×0.5 strength)     | 3.18*                                                          | 264*                   |
| <i>Nutrient rules — replenish nutrients when EC is too low</i>    |                                                    |                                      |                                                                |                        |
| R11                                                               | IF EC is <i>ok</i>                                 | → Nutrient: NONE                     | 0.00                                                           | 0                      |
| R12                                                               | IF EC is <i>high</i> or <i>very high</i>           | → Nutrient: NONE (over-concentrated) | 0.00                                                           | 0                      |
| R13                                                               | IF EC is <i>low</i>                                | → Nutrient: SMALL                    | 2.07                                                           | 172                    |
| R14                                                               | IF EC is <i>very low</i>                           | → Nutrient: MEDIUM (×0.8 strength)   | —                                                              | —                      |
| R15                                                               | IF EC is <i>very low</i>                           | → Nutrient: LARGE (×0.5 strength)    | 3.11*                                                          | 258*                   |
| <i>Compound safety rule — both inputs simultaneously critical</i> |                                                    |                                      |                                                                |                        |
| R16                                                               | IF pH is <i>very low</i> AND EC is <i>very low</i> | → pH-Up: MAX + Nutrient: LARGE       | (strength = $\min(\mu_{\text{pH\_vl}}, \mu_{\text{EC\_vl}})$ ) |                        |

\* Indicative centroid output for the Lettuce profile when the linguistic variable fires at full

membership ( $\mu = 1.0$ ). Where multiple output levels are activated simultaneously for the same valve (R4+R5, R9+R10, R14+R15), the centroid defuzzification aggregates all activated sets into a single crisp dose, producing smooth interpolation rather than a stepped response. Valve mutual exclusion (pH-Up and pH-Down never open simultaneously) is enforced in the defuzzification step, not at the rule level.

**Rule count clarification.** The rule base comprises **16 individual rule firings** (R1–R16 above), which correspond to **9 logical conditions** when grouped by their linguistic antecedent. Several conditions activate multiple output levels simultaneously with scaled firing strengths — for example, pH *very low* activates both LARGE (strength ×0.7) and MAX (strength ×0.5) on the pH-Up output (R4 and R5). This multi-level activation is a deliberate Mamdani design choice: it produces a graded, continuous dose curve through centroid defuzzification rather than a hard-switched output, improving dosing smoothness and reducing actuator wear.

## Defuzzification

The crisp valve open time is computed by the **centroid (centre of gravity)** defuzzification method implemented in `_centroid()`. For each point  $x$  in the output universe  $\mathcal{U}$ , the aggregated membership is the maximum over all activated output sets, each clipped (Mamdani clipping) to its firing strength  $\mu_i$ :

$$\mu_{\text{agg}}(x) = \max_i \left[ \min(\mu_i, \mu_{\text{out}_i}(x)) \right] \quad (3.4)$$

The crisp output  $t^*$  is then:

$$t^* = \frac{\sum_{x \in \mathcal{U}} x \cdot \mu_{\text{agg}}(x)}{\sum_{x \in \mathcal{U}} \mu_{\text{agg}}(x)} \quad (3.5)$$

If the denominator falls below  $10^{-9}$  (no rules activated),  $t^* = 0$  (no dosing).

### Safety Constraints

To ensure safe operation, several constraints are enforced:

- Mutual exclusion between pH-Up and pH-Down valves.
- Minimum activation threshold to avoid unnecessary wear.
- Maximum dosing limit per cycle to prevent over-correction.
- Blocking of nutrient dosing under high EC conditions.

### Validation Scenarios

The fuzzy inference engine was validated using the self-test suite embedded in `fuzzy.py`, executed against the Lettuce crop profile (flow rate: 83 mL/s). Table 3.6 presents the nine test scenarios, covering optimal conditions, single-variable pH and EC deviations, the compound safety rule, and a fuzzy-set boundary transition.

Table 3.6: Fuzzy inference engine validation results (Lettuce profile).

| pH  | EC  | pH-Up (s) | pH-Dn (s) | Nut (s) | Interpretation                                   |
|-----|-----|-----------|-----------|---------|--------------------------------------------------|
| 6.2 | 1.6 | 0.00      | 0.00      | 0.00    | Both optimal — no action (R1, R11)               |
| 5.8 | 1.6 | 0.00      | 0.00      | 0.00    | pH at low/ok boundary — below MIN_DOSE threshold |
| 5.2 | 1.6 | >0        | 0.00      | 0.00    | pH very low — pH-Up large/max (R4, R5)           |
| 6.8 | 1.6 | 0.00      | >0        | 0.00    | pH slightly high — pH-Down small (R8)            |
| 7.3 | 1.6 | 0.00      | >0        | 0.00    | pH very high — pH-Down medium/large (R9, R10)    |
| 6.2 | 0.5 | 0.00      | 0.00      | >0      | EC very low — Nutrient medium/large (R14, R15)   |
| 6.2 | 1.0 | 0.00      | 0.00      | >0      | EC slightly low — Nutrient small (R13)           |
| 6.2 | 2.5 | 0.00      | 0.00      | 0.00    | EC high — no nutrient dose (R12)                 |
| 5.1 | 0.4 | >0        | 0.00      | >0      | Both critical — compound rule R16 activated      |

All nine scenarios produced the expected valve decisions. The boundary case (pH = 5.8, EC = 1.6) confirmed that doses falling below the minimum threshold of 0.5s (41.5 mL) are suppressed, preventing unnecessary actuator wear. The compound scenario (pH = 5.1, EC = 0.4) triggered both the pH-Up and Nutrient valves simultaneously via R16, which fires at strength  $\min(\mu_{\text{pH\_vl}}, \mu_{\text{EC\_vl}})$ . Mutual exclusion between pH-Up and pH-Down was confirmed across all scenarios — no test produced non-zero output on both simultaneously.

## 1.7 Anomaly Detection — AI Disease Identification

The second intelligent component of the virtual twin is the LeafDiseaseCNN deep learning model, which provides an *anomaly detection* capability analogous to that found in industrial Digital Twin frameworks. Where industrial DTs use vibration or thermal signatures to detect equipment faults, VertiFarm uses visual leaf analysis to detect plant health anomalies.

The model was trained to classify plant images into 20 categories spanning three species — lettuce, tomato, and strawberry — covering bacterial infections, fungal diseases, viral conditions, and healthy states. Its integration into the Digital Twin follows the same bidirectional data pattern: the physical camera captures an image, the virtual model analyses it and produces a diagnosis, and the result is surfaced in the dashboard with a confidence score and top-5 probability ranking.

When a disease is detected, the virtual twin automatically generates a persistent Alert record in the database, creating an auditable trail of plant health events that contributes to the historical model.

## 1.8 Operator Interface — Dashboard as Digital Twin Window

The web-based dashboard serves as the human-facing window into the Digital Twin. It provides operators with a complete, real-time representation of the physical system's state across four specialised views:

- **Dashboard page** — live sensor values (pH, EC, temperature, humidity) with status classification and trend indicators, plus the Dosing Control panel showing the fuzzy controller's current decision and allowing mode switching between autonomous (fuzzy) and manual operation.
- **Analytics page** — interactive time-series charts of historical measurements retrieved from the database via the `/api/history` endpoint, enabling retrospective analysis of system behaviour over selectable time windows (1 hour to 7 days).
- **Detection page** — the AI plant disease scanning interface, providing a live camera preview and on-demand inference with result visualisation including the top-5 classification probabilities.
- **Settings page** — configuration of the virtual model's behavioural parameters: active crop selection (which reconfigures the fuzzy membership functions), dosing timing intervals, and account management.

This separation of concerns — live state, history, intelligent inference, and configuration — directly mirrors the four-layer structure of a Digital Twin described by Tao et al. [66]: *physical data, virtual model, service, and connection*.

## 1.9 Related Work on Digital Twins in Hydroponics

The application of Digital Twin technology to hydroponic and controlled-environment agriculture has attracted growing research interest over the last five years. Early DT implementations were primarily *monitoring twins*, focusing on data collection and visualisation with no autonomous control capability [68]. More advanced systems integrated plant growth models: Alinejad et al. [67] coupled sensor data with a Penman–Monteith model to reduce water consumption by 28 %, while Chaux et al. [73] proposed a physics-based nutrient uptake model for NFT systems, though both remained computationally intensive or crop-specific. On the machine learning side, Bersani et al. [74] used LSTM networks to predict pH and EC trajectories up to 30 minutes ahead, yet their platform required persistent cloud connectivity. Computer vision-based disease detection achieved accuracies of 85–99 % [75], but classifiers remained disconnected from the DT control loop. Fuzzy logic approaches improved pH stability by up to 40 % [69], though embedded deployment remained limited [76]. Across all reviewed works, five recurring gaps emerge: cloud dependency, separation of monitoring and control, absence of AI integration in the control loop, lack of crop-adaptive behavioural models, and no unified locally-hosted operator interface combining live state, analytics, and AI inference.

## 1.10 Comparison of VertiFarm with Related Digital Twin Works

Table 3.7 provides a systematic comparison of VertiFarm against selected representative works from the literature across eight Digital Twin capability dimensions.

Table 3.7: Comparative analysis of VertiFarm against related hydroponic Digital Twin implementations.

| Capability                     |  | Verdouw<br>et al. [68] | Bersani<br>et al. [74] | Srudys<br>et al. [69] | Chaux<br>et al. [73] | VertiFarm<br>(ours) |
|--------------------------------|--|------------------------|------------------------|-----------------------|----------------------|---------------------|
| Real-time sensor monitoring    |  | Yes                    | Yes                    | Yes                   | Yes                  | Yes                 |
| Persistent historical database |  | Yes                    | Yes                    | No                    | Yes                  | Yes                 |
| Autonomous closed-loop control |  | No                     | ~                      | Yes                   | ~                    | Yes                 |
| Fuzzy logic behavioural model  |  | No                     | No                     | Yes                   | No                   | Yes                 |
| AI plant disease detection     |  | No                     | No                     | No                    | No                   | Yes                 |
| Multi-crop adaptive model      |  | No                     | No                     | No                    | No                   | Yes                 |
| Unified web operator interface |  | ~                      | Yes                    | No                    | ~                    | Yes                 |
| Edge/offline deployment        |  | No                     | No                     | ~                     | No                   | Yes                 |

Yes = Fully implemented
~ = Partially implemented
No = Not implemented

The comparison reveals that VertiFarm is the only implementation in the reviewed literature that simultaneously achieves all five core Digital Twin characteristics — real-time monitoring, persistent history, autonomous closed-loop control, AI anomaly detection, and a unified operator interface — within a single locally-hosted, edge-deployable architecture. The key differentiating contributions relative to the state of the art are:

- **Integrated AI disease detection:** unlike all reviewed systems, VertiFarm embeds the LeafDiseaseCNN classifier directly into the DT loop, with detected anomalies automatically persisted as database alerts and surfaced in the operator dashboard.
- **Multi-crop fuzzy model:** the crop-adaptive architecture — selectable between Lettuce, Tomato, and Strawberry via the Settings page — dynamically reconfigures all fuzzy membership function parameters, enabling a single physical installation to serve different cultivation scenarios without code modification.
- **Pure-Python fuzzy engine:** unlike the implementation of Castañeda-Miranda et al. [76] which depends on `scikit-fuzzy`, the VertiFarm fuzzy engine is implemented entirely

in standard Python, eliminating library dependencies and enabling deployment on constrained hardware such as the Raspberry Pi 5.

- **Edge-first architecture:** the entire DT stack — sensor acquisition, database, fuzzy controller, AI inference, and web dashboard — runs on a single Raspberry Pi 5 with no cloud dependency, ensuring uninterrupted operation in environments with limited or no internet connectivity.

## 2 HydroBot: A Domain-Constrained Conversational Assistant for the Hydroponic System

### 2.1 Motivation and Design Objectives

The VertiFarm platform integrates three distinct AI layers: a fuzzy logic controller for autonomous chemical dosing, a convolutional neural network for plant disease detection, and the conversational assistant described in this section. While the two former components operate autonomously and produce structured, machine-readable outputs, a fundamental gap remains between raw system data and actionable agronomic knowledge for the human operator. Sensor readings, dosing decisions, and disease detection confidence scores are meaningful only when their interpretation is accessible without requiring expertise in data science or control engineering.

A natural language interface bridges this gap by allowing operators to interrogate the system in plain English, receiving contextualised, expert-level guidance without consulting external documentation. The conversational assistant — referred to as **HydroBot** throughout this section — was therefore designed with three explicit objectives:

1. **Domain restriction:** responses must be confined to hydroponics, plant science, and system operation. Off-topic queries must be refused before they reach the language model, preventing misuse and ensuring output reliability.
2. **Offline operation:** the inference engine must run entirely on the Raspberry Pi 5 without any external network dependency, preserving system autonomy in isolated or bandwidth-limited deployments.
3. **Platform integration:** HydroBot must be embedded directly into the VertiFarm dashboard as a first-class component, sharing the same authentication, routing, and visual design system as the sensor monitoring, analytics, and disease detection pages.

### 2.2 System Architecture

Unlike the standalone chatbot prototypes common in the literature, HydroBot is not a separate microservice. It is implemented as an additional route group within the main **FastAPI** application (`main.py`), inheriting the platform's existing session-based authentication, database connection pool, and static file serving infrastructure. This monolithic integration eliminates inter-service communication overhead and simplifies deployment on resource-constrained hardware.

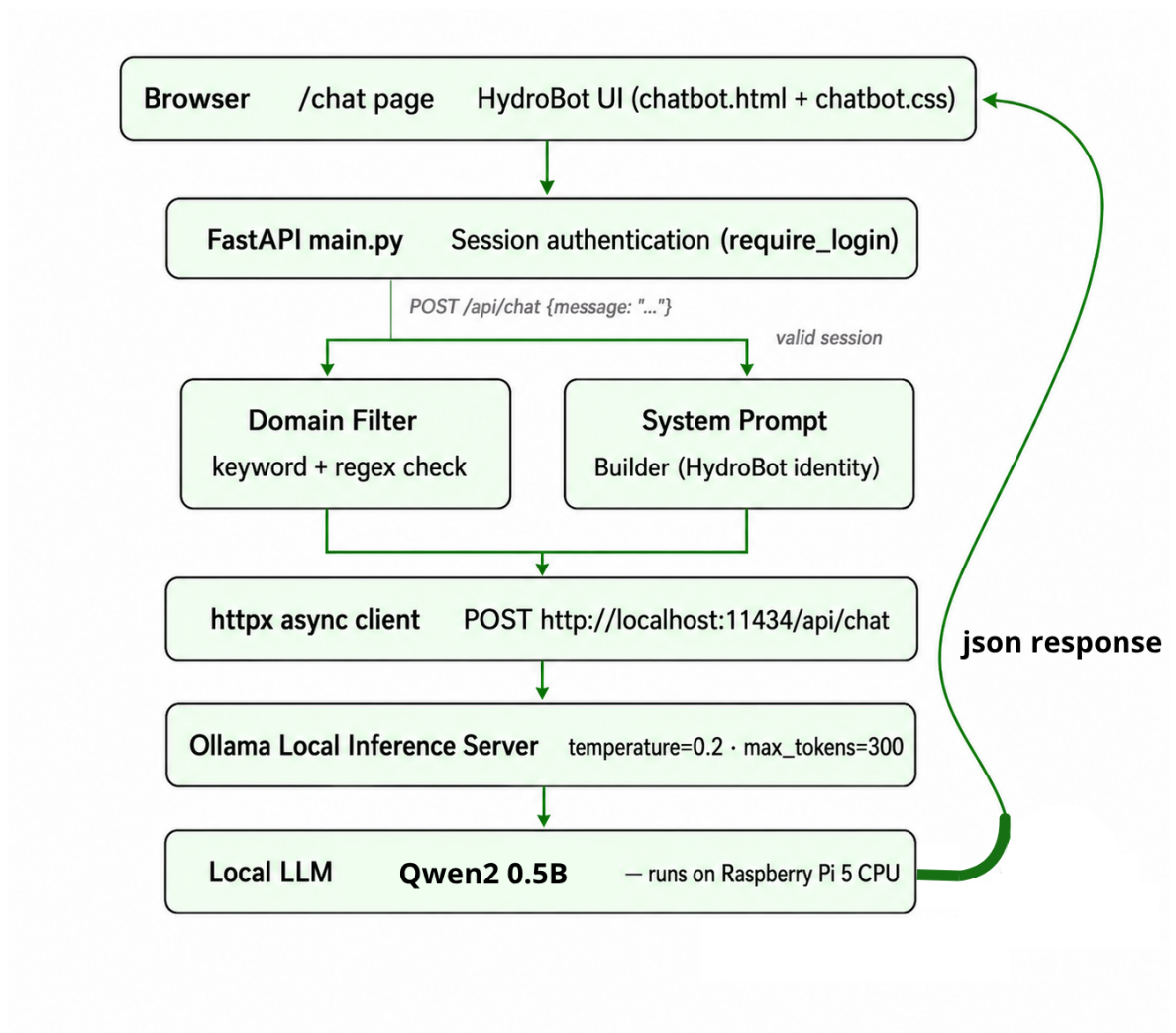


Figure 3.6: End-to-end architecture of the local AI chatbot system integrated into the VertiFarm platform.

The two server-side endpoints added to `main.py` are:

- **GET /chat** — returns `chatbot.html`, the dedicated chat interface page. Access is gated by the same session cookie that protects all other dashboard pages.
- **POST /api/chat** — the inference proxy endpoint protected by `Depends(require_login)`. It receives a JSON body `{message: str}`, applies the domain filter, forwards valid requests to Ollama, and returns `{answer: str}`.

## 2.3 Domain Filtering and Safety Mechanism

Deploying a general-purpose language model in an industrial monitoring context carries the risk of the model producing responses on arbitrary topics, potentially misleading operators with unrelated or harmful information. To mitigate this risk, a three-stage domain filter is applied to every incoming message *before* the language model is invoked. This design ensures that off-topic requests are rejected with zero inference cost.

## Blocked Topic Detection

The first stage checks whether the message contains any token from a predefined list of categories that are explicitly outside the system’s scope, including: politics, religion, entertainment, hacking, and violence. A match at this stage immediately terminates the request pipeline and returns a standardised refusal response to the client.

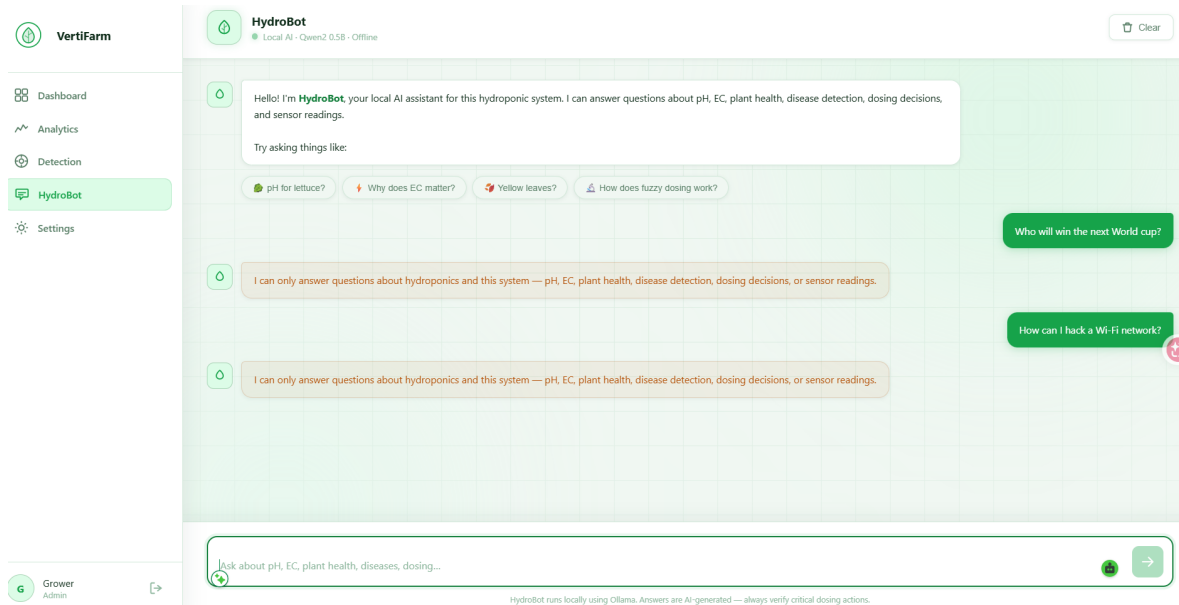


Figure 3.7: Blocked-topic detection in HydroBot.

Figure 3.7 illustrates the behaviour of the first filtering stage when an operator submits a query outside the hydroponic domain. The request is intercepted before any LLM inference occurs, ensuring both safety and zero inference cost.

## Hydroponic Keyword Allowlist

The second stage checks the message against a vocabulary of domain-specific terms. The allowlist encompasses the key concepts relevant to the VertiFarm system, including chemical parameters (pH, EC, TDS, alkaline, acidic), plant biology (nutrient, disease, deficiency, root, leaf, yield), sensor types (temperature, humidity), crops (lettuce, tomato, strawberry), and system components (valve, pump, actuator, fuzzy, dosing, reservoir). A match at this stage permits the request to proceed.

## Semantic Pattern Matching

The third stage applies a set of compiled regular-expression patterns to capture diagnostic phrasings that would not match individual keywords but are unmistakably system-relevant, such as: “*Why did the valve activate?*”, “*What should I check?*”, and “*How much nutrient should I add?*”. This stage prevents operators from inadvertently having their legitimate queries blocked by an overly restrictive keyword filter.

The complete filtering logic is formalised in Algorithm 1.

---

**Algorithm 1: HydroBot Domain Filter**

---

**Input:** User message  $m$   
**Output:** ALLOWED | BLOCKED

```
1 Normalise  $m$  to lowercase and strip whitespace;  
2 foreach  $t \in \text{BLOCKEDTOPICS}$  do  
3   | if  $t \in m$  then  
4   |   | return BLOCKED;  
  
5 foreach  $k \in \text{HYDROPONICKEYWORDS}$  do  
6   | if  $k \in m$  then  
7   |   | return ALLOWED;  
  
8 foreach  $p \in \text{SEMANTICPATTERNS}$  do  
9   | if regex.search( $p$ ,  $m$ ) then  
10  |   | return ALLOWED;  
11 return BLOCKED;
```

---

## 2.4 System Prompt Engineering

For messages that pass the domain filter, a structured system prompt is prepended to the inference request. System prompting — the practice of providing the model with a persistent instruction context before the user message — is a well-established technique for constraining the behaviour of instruction-tuned language models without modifying their weights [77,78].

The HydroBot system prompt defines six behavioural constraints:

1. The model must identify itself as *HydroBot*, a specialised assistant embedded in the VertiFarm platform.
2. It must restrict responses to hydroponics, nutrient chemistry, sensor interpretation, plant disease, and system operation.
3. Responses must be practical, concise, and actionable (3–5 sentences maximum).
4. Sensor values, timestamps, and detection results must never be fabricated — if real values are needed, the user must be directed to the dashboard.
5. For any dosing or hardware action, the model must recommend sensor verification before proceeding.
6. Uncertainty must be explicitly acknowledged rather than concealed with confident but incorrect statements.

The inference parameters are deliberately conservative: a temperature of 0.2 suppresses creative variation in favour of reproducible, factual responses, and the maximum token limit of 300 enforces concise answers appropriate for a monitoring dashboard context.

## 2.5 Offline LLM Inference with Ollama

A fundamental constraint of the production deployment target — a Raspberry Pi 5 operating in a hydroponic installation that may have limited or no internet access — precludes the use of cloud-hosted inference APIs. **Ollama** [79] was adopted as the local inference framework for three reasons. First, it exposes an HTTP REST endpoint on `localhost:11434` that is structurally identical to the OpenAI Chat Completions API, requiring minimal code changes

during development. Second, it supports GGUF-quantised model formats, substantially reducing memory footprint and enabling deployment on hardware with 4–8 GB of RAM. Third, Ollama installs as a persistent background service, requiring no operator intervention to start the inference engine at system boot.

Table 3.8 compares the candidate models evaluated for deployment on the Raspberry Pi 5.

Table 3.8: Evaluation of local LLM candidates for Raspberry Pi 5 deployment under the VertiFarm memory budget.

| Model               | Parameters  | Quantisation | RAM (approx.) | Avg. response latency (Pi 5) |
|---------------------|-------------|--------------|---------------|------------------------------|
| <b>Qwen2 0.5B</b>   | <b>0.5B</b> | <b>Q4_K</b>  | ~0.4 GB       | ~5–10 s                      |
| <b>Phi-3 Mini</b>   | 3.8B        | Q4_K         | ~2.3 GB       | ~20–35 s                     |
| <b>LLaMA 3.2 3B</b> | 3.2B        | Q4_K         | ~2.0 GB       | ~25–40 s                     |
| <b>Mistral 7B</b>   | 7B          | Q4_0         | ~4.5 GB       | ~60–90 s                     |

The Raspberry Pi 5 used in this project is equipped with 8 GB of LPDDR4X RAM, of which a portion is reserved by the operating system, the sensor background loop, the SQLite database engine, and the LeafDiseaseCNN model already loaded in memory. Given this shared memory budget, **Qwen2 0.5B** was selected as the default model. Its lightweight architecture significantly reduces RAM consumption and inference latency, making it well-suited for real-time execution on the Raspberry Pi 5 while maintaining sufficient performance for closed-domain hydroponic question answering tasks.

It is important to note that the LLM inference runs on the **CPU** of the Raspberry Pi 5, since no compatible GPU accelerator is attached. This accounts for the latency figures reported in Table 3.8. In deployments where a co-processor is available (e.g., a Hailo AI accelerator), inference latency can be expected to decrease significantly.

## 2.6 User Interface and Operator Interaction

The HydroBot interface is implemented as a dedicated page (/chat) within the VertiFarm dashboard, styled consistently with the botanical light theme applied across all four dashboard pages. The interface provides:

- **Conversation history** rendered in a scrollable message thread, with user messages displayed in green bubbles on the right and HydroBot responses in white cards on the left, each preceded by a leaf-motif avatar.
- **Animated typing indicator** (three bouncing dots) displayed while the inference request is in flight, providing visual feedback during the 20–35 second model response latency.
- **Suggestion chips** on the welcome message offering one-click example queries covering the four primary use cases: nutrient chemistry, disease detection interpretation, system state, and fuzzy controller decisions.
- **Error states** that distinguish between a blocked off-topic message (amber) and an Ollama connectivity failure (red), with an actionable error message in the latter case.

- **Auto-resizing textarea** that accommodates multi-line questions, with Enter to send and Shift+Enter for a newline, consistent with contemporary chat interface conventions.

Representative examples of operator interactions enabled by HydroBot are presented in Table 3.9.

Table 3.9: Representative operator queries and HydroBot response scope.

| Operator query                                                            | HydroBot response scope                                                                                              |
|---------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------|
| “The EC is 3.2 mS/cm; is that safe for lettuce?”                          | Interprets value against optimal range (1.2–2.0), identifies over-concentration risk, suggests dilution.             |
| “Why did the nutrient valve open this cycle?”                             | Explains fuzzy logic EC-low rule, references dosing volume and mixing wait interval.                                 |
| “The model detected early blight with 74% confidence — what should I do?” | Describes early blight symptoms, recommends removal of affected leaves, improved air circulation, fungicide options. |
| “What is the ideal pH for the vegetative stage of lettuce?”               | Provides agronomic target range (5.8–6.5), explains effect of pH on nutrient availability at each band.              |
| “The plant leaves are turning yellow at the edges.”                       | Differential diagnosis: calcium deficiency, pH imbalance, or over-watering; recommends sensor check sequence.        |

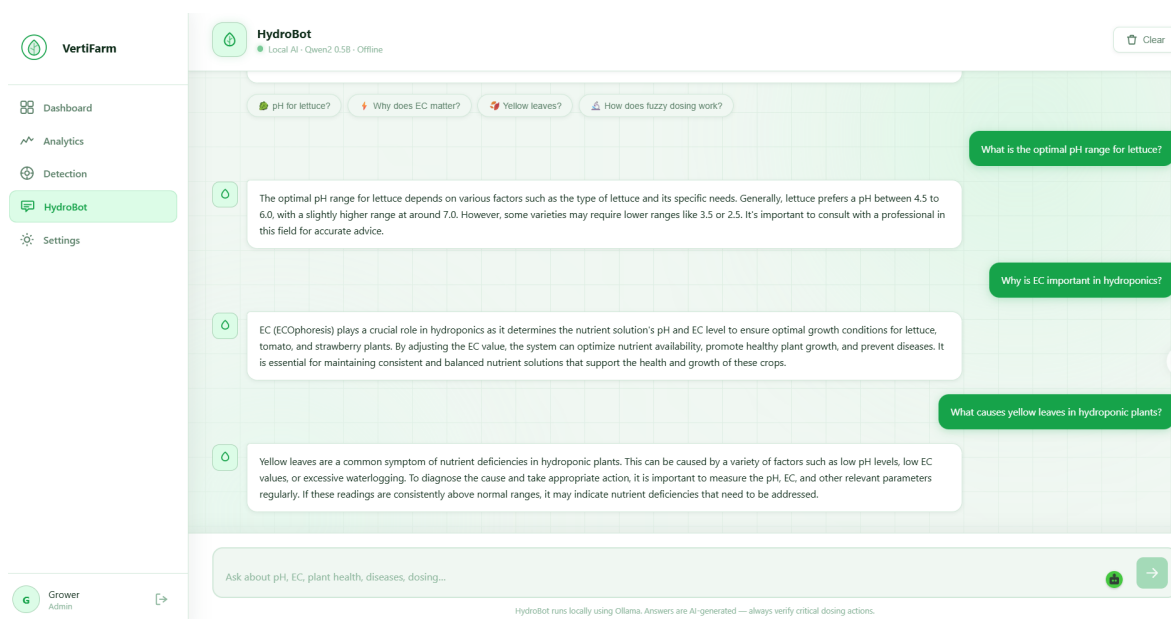


Figure 3.8: HydroBot chatbot interface integrated into the VertiFarm dashboard.

## Conclusion

This chapter presented the three intelligent components constituting the VertiFarm automation and supervision framework: the fuzzy logic dosing controller, the digital twin, and the HydroBot conversational assistant.

The fuzzy logic controller autonomously regulates nutrient concentration and pH by mapping sensor readings to actuator commands through agronomic expert rules, ensuring optimal growing conditions without manual intervention.

The digital twin provides real-time synchronisation between the physical tower and its virtual representation, maintains a complete behavioural history, and exposes dynamic reconfiguration through the Settings page, forming a complete single-site Digital Twin architecture at laboratory scale.

HydroBot completes the system by providing a contextual decision-support layer: domain filtering, structured prompt engineering, and local inference on the Raspberry Pi 5 via Ollama, ensuring intelligent, low-latency, and network-independent interaction.

Together with the LeafDiseaseCNN disease detection module presented in the previous chapter, these components turn VertiFarm into a fully autonomous and adaptive cyber-physical system.

## GENERAL CONCLUSION

This thesis presented the design and implementation of an intelligent hydroponic system integrating Artificial Intelligence, IoT-based monitoring, fuzzy logic control, digital twin modelling, and a conversational assistant. The main objective of this work was to propose a smart and modular platform capable of supporting hydroponic cultivation through automatic monitoring, plant disease detection, crop yield estimation, nutrient regulation, and user-friendly interaction.

The first chapter established the theoretical foundations required for the development of the proposed system. It introduced the main concepts of Artificial Intelligence, Machine Learning, and Deep Learning, with a particular focus on Convolutional Neural Networks and their role in image-based agricultural applications. Transfer learning architectures, including ResNet-50 and EfficientNet-B3, were also presented as efficient solutions for plant disease classification. In addition, the chapter discussed object detection and segmentation using YOLO models, evaluation metrics such as accuracy, precision, recall, F1-score, IoU, and mAP, as well as fuzzy logic and its importance for decision-making under uncertainty. These concepts formed the scientific basis for the practical developments presented in the following chapters.

The second chapter focused on the design, training, and evaluation of the proposed Artificial Intelligence models. A plant disease detection model was developed using three deep learning architectures: a Custom CNN, ResNet-50, and EfficientNet-B3. These models were trained and evaluated on a heterogeneous dataset containing 20 classes related to tomato, lettuce, and strawberry plants. The comparative analysis showed that the three models achieved strong classification performance, confirming the relevance of deep learning for automatic disease diagnosis in controlled agricultural environments. Furthermore, a YOLO11-based instance segmentation model was implemented for tomato detection and ripeness classification. This model enabled the localization of individual tomatoes, the identification of their maturity stage, and the estimation of crop yield using predefined average weights. The obtained results demonstrated the potential of computer vision for both plant health monitoring and harvest estimation.

The third chapter presented the integration of the intelligent modules into the proposed hydroponic platform. A digital twin architecture was developed to represent the physical hydroponic system and provide real-time visualization of its state. This digital twin integrates sensor data, system monitoring, fuzzy logic control, and AI-based diagnosis, allowing the user to supervise the system more effectively. The Mamdani fuzzy logic controller was designed to regulate nutrient dosing and pH correction according to sensor values and expert-defined rules. This approach allowed the system to handle uncertain and variable conditions in a flexible manner. In addition, HydroBot, a domain-constrained conversational assistant, was integrated into the platform to facilitate interaction between the user and the system. It provides an intuitive interface for asking questions, understanding system status, and supporting decision-making.

Overall, the proposed system demonstrates the feasibility and relevance of combining Artificial Intelligence, fuzzy logic, IoT, digital twin technology, and conversational AI within a single smart hydroponic platform. The integration of these components contributes to improving crop monitoring, automating control decisions, reducing manual intervention, and supporting sustainable agriculture. This work also highlights the importance of developing intelligent agricultural systems that are not limited to data visualization, but are able to analyze, decide, predict, and interact with users.

However, several improvements can be considered for future work. First, the system should be tested under real greenhouse or vertical farming conditions over a longer period in order to evaluate its robustness, reliability, and practical performance. Second, the plant disease dataset can be extended with more real hydroponic images, additional plant species, and more disease classes to improve model generalization. Third, the yield estimation model can be enhanced by incorporating more fruit types, more growth stages, and real weight measurements. Moreover, the fuzzy controller can be further improved by adding adaptive rules or optimization techniques to make the control process more dynamic. Finally, the digital twin can be extended with predictive simulation capabilities, allowing the system to anticipate future plant growth, nutrient needs, and possible anomalies.

In conclusion, this thesis represents a step toward the development of an intelligent, autonomous, and sustainable hydroponic system. By combining advanced AI models, fuzzy control, digital twin modelling, and a conversational assistant, the proposed platform offers a promising solution for smart agriculture and opens new perspectives for future research in precision farming and controlled-environment agriculture.

## BIBLIOGRAPHY

- [1] G. Jocher and J. Qiu, "Ultralytics yolo11," 2024.
- [2] KDnuggets, "The ultimate scikit-learn machine learning cheatsheet." <https://www.kdnuggets.com/2021/01/ultimate-scikit-learn-machine-learning-cheatsheet.html>, 2021. Accessed: 2026-06-03.
- [3] HyperAI, "Confusion matrix." <https://hyper.ai/fr/wiki/29504>, 2026. Accessed: 2026-06-03.
- [4] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.
- [5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, vol. 25, pp. 1097–1105, 2012.
- [6] S. P. Mohanty, D. P. Hughes, and M. Salathé, "Using deep learning for image-based plant disease detection," *Frontiers in Plant Science*, vol. 7, p. 1419, 2016.
- [7] E. C. Too, L. Yujian, S. Njuki, and L. Yingchun, "A comparative study of fine-tuning deep learning models for plant disease identification," *Computers and Electronics in Agriculture*, vol. 161, pp. 272–279, 2019.
- [8] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *arXiv preprint arXiv:1506.02640*, 2016.
- [9] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.
- [10] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," *arXiv preprint arXiv:2004.10934*, 2020.
- [11] R. Khanam, M. Hussain, R. Hill, and P. Allen, "A comprehensive review of convolutional neural networks for defect detection in industrial applications," *IEEE Access*, 2024.
- [12] S. Mallick, "Yolo11 - learnopencv," 2024.
- [13] J. Feng, Q. An, J. Zhang, S. Zhou, G. Du, and K. Yang, "Application of yolov7-tiny in the detection of steel surface defects," in *2024 5th International Seminar on Artificial Intelligence, Networking and Information Technology (AINIT)*, pp. 2241–2245, IEEE, 2024.
- [14] M. S. Mir, N. B. Naikoo, R. H. Kanth, F. A. Bahar, M. A. Bhat, A. Nazir, S. S. Mahdi, Z. Amin, L. Singh, W. Raja, and A. A. Saad, "Vertical farming: The future of agriculture: A review," *The Pharma Innovation Journal*, vol. 11, no. 2, pp. 1175–1195, 2022.

- [15] S. N. Bhanja, A. Mukherjee, and M. Rodell, "Groundwater storage change detection from in situ and grace-based estimates in major river basins across india," *Hydrological Sciences Journal*, vol. 65, no. 4, pp. 650–659, 2020.
- [16] Y. Kobayashi, T. Kotilainen, G. Carmona-Garcia, A. Leip, and H. L. Tuomisto, "Vertical farming: A trade-off between land area needs for crops and for renewable energy production," *Journal of Cleaner Production*, vol. 379, p. 134507, 2022.
- [17] M. Rufi-Salis, M. J. Calvo, A. Petit-Boix, G. Villalba, and X. Gabarrell, "Exploring nutrient recovery from hydroponics in urban agriculture: An environmental assessment," *Resources, Conservation and Recycling*, vol. 155, p. 104683, 2020.
- [18] S. Eshkabilov, A. Lee, X. Sun, C. W. Lee, and H. Simsek, "Hyperspectral imaging techniques for rapid detection of nutrient content of hydroponically grown lettuce cultivars," *Computers and Electronics in Agriculture*, vol. 181, p. 105968, 2021.
- [19] J. W. Yang and H. I. Kim, "An overview of recent advances in greenhouse strawberry cultivation using deep learning techniques: A review for strawberry practitioners," *Agronomy*, vol. 14, no. 1, p. 34, 2024.
- [20] R. Concepcion, S. Lauguico, E. Dadios, A. Bandala, E. Sybingco, and J. Alejandrino, "Tomato septoria leaf spot necrotic and chlorotic regions computational assessment using artificial bee colony-optimized leaf disease index," in *2020 IEEE Region 10 Conference*, pp. 1243–1248, IEEE, 2020.
- [21] R. S. Concepcion, S. C. Lauguico, R. R. Tobias, E. P. Dadios, A. A. Bandala, and E. Sybingco, "Estimation of photosynthetic growth signature at the canopy scale using new genetic algorithm-modified visible band triangular greenness index," in *2020 International Conference on Advanced Robotics and Intelligent Systems*, pp. 1–6, IEEE, 2020.
- [22] P. Koohakan, T. Jeanaksorn, and I. Nuntagij, "Major diseases of lettuce grown by commercial nutrient film technique in thailand," *Current Applied Science and Technology*, vol. 8, no. 2, pp. 56–63, 2008.
- [23] Cornell University, "Downy mildew on lettuce," 2024. Available online: <https://blogs.cornell.edu/livepath/gallery/lettuce/downy-mildew-on-lettuce/>.
- [24] V. K. Vishnoi, K. Kumar, and B. Kumar, "Plant disease detection using computational intelligence and image processing," *Journal of Plant Diseases and Protection*, vol. 128, pp. 19–53, 2021.
- [25] V. K. Vishnoi, K. Kumar, and B. Kumar, "A comprehensive study of feature extraction techniques for plant leaf disease detection," *Multimedia Tools and Applications*, vol. 81, no. 1, pp. 367–419, 2022.
- [26] P. Sajitha *et al.*, "A review on machine learning and deep learning image-based plant disease detection and classification," *Smart Agricultural Technology*, 2024.
- [27] I. Pacal *et al.*, "A systematic review of deep learning techniques for plant disease detection," *Artificial Intelligence Review*, 2024.
- [28] S. Mustofa, M. M. H. Munna, Y. R. Emon, G. Rabbany, and M. T. Ahad, "A comprehensive review on plant leaf disease detection using deep learning," *arXiv preprint arXiv:2308.14087*, 2023.
- [29] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

- [30] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.
- [31] W. Wang, Y. Li, T. Zou, X. Wang, J. You, and Y. Luo, "A novel image classification approach via dense-mobilenet models," *Mobile Information Systems*, vol. 2020, p. 7602384, 2020.
- [32] W. Deng, F. Zhou, Z. Gong, Y. Cui, L. Liu, and Q. Chi, "Disease feature recognition of hydroponic lettuce images based on support vector machine," *Traitement du Signal*, vol. 39, no. 2, 2022.
- [33] H. Aquino, E. Sybingco, R. Mendigoria, A. Bandala, O. J. Alajas, E. Dadios, and R. R. Vicerra, "On-demand healthy and chlorotic lactuca sativa leaf classification using svm in a rotating hydroponic system," in *2022 IEEE 14th International Conference on Humanoid, Nanotechnology, IT, Communication and Control, Environment, and Management*, pp. 1–5, IEEE, 2022.
- [34] I. Y. Pratama, A. Wahab, and M. Alaydrus, "Deep learning for assessing unhealthy lettuce hydroponic using cnn based on faster r-cnn with inception v2," in *2020 Fifth International Conference on Informatics and Computing*, pp. 1–6, IEEE, 2020.
- [35] B. Bryan, A. G. M. Don, A. A. Gotengco, E. V. Millena, M. G. P. Beaño, E. A. Mandayo, O. A. Medina, and A. L. F. Sigue, "Lettuce root development and monitoring system using ml in hydroponics," in *TENCON 2022*, pp. 1–6, IEEE, 2022.
- [36] A. S. Rathor, S. Choudhury, A. Sharma, P. Nautiyal, and G. Shah, "A novel deep convolutional neural network for efficient classification of lettuce diseases," *Procedia Computer Science*, vol. 258, pp. 755–764, 2025.
- [37] X. Zhang, D. Cao, M. Wang, G. Cui, Y. Shi, and Y. Cui, "Nutrient deficiency diagnosis in whole hydroponic lettuce based on random forest," *INMATEH-Agricultural Engineering*, vol. 68, no. 3, 2022.
- [38] Z. Gul and S. Bora, "Exploiting pre-trained cnn for the detection of nutrient deficiencies in hydroponic basil," *Sensors*, vol. 23, no. 12, p. 5407, 2023.
- [39] Z. Xu, X. Guo, A. Zhu, X. He, X. Zhao, Y. Han, and R. Subedi, "Using deep convolutional neural networks for image-based diagnosis of nutrient deficiencies in rice," 2020.
- [40] T. Anandhakrishnan *et al.*, "Deep convolutional neural networks for image based tomato leaf disease detection," *Sustainable Chemistry and Pharmacy*, vol. 30, p. 100793, 2022.
- [41] H. Rehana, M. Ibrahim, and M. H. Ali, "Plant disease detection using region-based convolutional neural network," *arXiv preprint arXiv:2303.09063*, 2023.
- [42] M. Thalor *et al.*, "Performance comparison of cnn models for tomato disease detection using image-based data in both controlled and real-world conditions," *Current Agriculture Research Journal*, 2024.
- [43] J. Liu and X. Wang, "Tomato disease object detection method combining prior knowledge attention mechanism and multiscale features," *Frontiers in Plant Science*, vol. 14, p. 1255119, 2023.
- [44] X. Wang *et al.*, "An efficient deep learning model for tomato disease detection," *Plant Methods*, 2024.

- [45] M. Jelali *et al.*, "Deep learning networks-based tomato disease and pest detection in agricultural systems: A review," *Frontiers in Plant Science*, 2024.
- [46] U. Afzaal, B. Bhattarai, Y. R. Pandeya, and J. Lee, "An instance segmentation model for strawberry diseases based on mask r-cnn," *Sensors*, vol. 21, no. 19, p. 6565, 2021.
- [47] U. Afzaal, "Strawberry disease detection dataset," 2021. Available online on Kaggle.
- [48] S. Karki, J. K. Basak, N. Tamrakar, N. C. Deb, B. Paudel, J. H. Kook, M. Y. Kang, D. Y. Kang, and H. T. Kim, "Strawberry disease detection using transfer learning of deep convolutional neural networks," *Scientia Horticulturae*, vol. 332, p. 113241, 2024.
- [49] A. Musa, M. Hassan, M. Hamada, and F. Aliyu, "Low-power deep learning model for plant disease detection for smart-hydroponics using knowledge distillation techniques," *Journal of Low Power Electronics and Applications*, vol. 12, no. 2, p. 24, 2022.
- [50] A. J. Hati and R. R. Singh, "Ai-driven pheno-parenting: A deep learning-based plant phenotyping trait analysis model on a novel soilless farming dataset," *IEEE Access*, vol. 11, pp. 35298–35314, 2023.
- [51] T. Y. Lee, I. A. Lin, J. Y. Yu, J. M. Yang, and Y. C. Chang, "High efficiency disease detection for potato leaf with convolutional neural network," *SN Computer Science*, vol. 2, no. 4, p. 297, 2021.
- [52] J. Amara, B. Bouaziz, and A. Algergawy, "A deep learning-based approach for banana leaf diseases classification," 2017.
- [53] A. Waheed, M. Goyal, D. Gupta, A. Khanna, A. E. Hassanien, and H. M. Pandey, "An optimized dense convolutional neural network model for disease recognition and classification in corn leaf," *Computers and Electronics in Agriculture*, vol. 175, p. 105456, 2020.
- [54] V. K. Vishnoi, K. Kumar, B. Kumar, S. Mohan, and A. A. Khan, "Detection of apple plant diseases using leaf images through convolutional neural network," *IEEE Access*, vol. 11, pp. 6594–6609, 2022.
- [55] Kaggle, "Lettuce plant disease dataset," 2024. Available online: <https://www.kaggle.com/datasets/santoshshaha/lettuce-plant-disease-dataset>.
- [56] I. Kandel and M. Castelli, "The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset," *ICT Express*, vol. 6, no. 4, pp. 312–315, 2020.
- [57] D. P. Hughes and M. Salathé, "An open access repository of images on plant health to enable the development of mobile disease diagnostics," 2015.
- [58] U. Afzaal, B. Bhattarai, Y. R. Pandeya, and J. Lee, "An instance segmentation model for strawberry diseases based on Mask R-CNN," *Sensors*, vol. 21, no. 19, p. 6565, 2021.
- [59] S. Shah, "Lettuce plant disease dataset." <https://www.kaggle.com/datasets/santoshshaha/lettuce-plant-disease-dataset>, 2023.
- [60] A. Buslaev, V. I. Iglovikov, E. Khvedchenya, A. Parinov, M. Druzhinin, and A. A. Kalinin, "Albumentations: Fast and flexible image augmentations," *Information*, vol. 11, no. 2, p. 125, 2020.

- [61] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *Proceedings of the 36th International Conference on Machine Learning*, pp. 6105–6114, 2019.
- [62] M. Agarwal, S. K. Gupta, and K. K. Biswas, "Development of efficient cnn model for tomato crop disease identification," *Sustainable Computing: Informatics and Systems*, vol. 28, p. 100407, 2020.
- [63] G. Geetharamani and J. Arun Pandian, "Identification of plant leaf diseases using a nine-layer deep convolutional neural network," *Computers and Electrical Engineering*, vol. 76, pp. 323–338, 2019.
- [64] Laboro.AI, "Laboro tomato dataset." Roboflow Universe, 2021.
- [65] M. Grieves, "Digital twin: Manufacturing excellence through virtual factory replication," *White Paper*, 2014. Florida Institute of Technology.
- [66] F. Tao, H. Zhang, A. Liu, and A. Y. C. Nee, "Digital twin in industry: State-of-the-art," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 4, pp. 2405–2415, 2019.
- [67] M. Alinejad, M. Mokarram, A. Khaksari, H. R. Pourghasemi, and J. P. Tiefenbacher, "IoT-based irrigation management using machine learning and precision agriculture for greenhouse crops," *Computers and Electronics in Agriculture*, vol. 181, p. 105950, 2021.
- [68] C. Verdouw, B. Tekinerdogan, A. Beulens, and S. Wolfert, "Digital twins in smart farming," *Agricultural Systems*, vol. 189, p. 103046, 2021.
- [69] L. Srudys, B. R. Gerónimo, A. Pacheco-Olvera, R. Fuentes-Pérez, and L. A. Morales-Hernandez, "Fuzzy logic control system for hydroponic nutrient solution monitoring," *Applied Sciences*, vol. 9, no. 3, p. 399, 2019.
- [70] H. M. Resh, *Hydroponic Food Production: A Definitive Guidebook for the Advanced Home Gardener and the Commercial Hydroponic Grower*. Boca Raton, FL: CRC Press, 8th ed., 2022.
- [71] D. R. Hoagland and D. I. Arnon, *The Water-Culture Method for Growing Plants Without Soil*. Circular 347, Berkeley, CA: California Agricultural Experiment Station, 1950.
- [72] Food and Agriculture Organization of the United Nations, *Hydroponic Crop Production in Greenhouses*. Rome, Italy: FAO, 2003.
- [73] J. D. Chaux, L. A. Alvarez-Vaneg, and I. Sánchez-Suárez, "Integrating IoT and digital twin technology in hydroponic systems for monitoring and control," *Electronics*, vol. 10, no. 24, p. 3106, 2021.
- [74] C. Bersani, C. Ruggiero, A. M. Tomasoni, E. Trasforini, and P. Bellavista, "IoT approach for smart management of a vertical farm using a digital twin," *IEEE Internet of Things Journal*, vol. 7, no. 6, pp. 5544–5554, 2020.
- [75] A. Kamilaris and F. X. Prenafeta-Boldú, "Deep learning in agriculture: A survey," *Computers and Electronics in Agriculture*, vol. 147, pp. 70–90, 2018.
- [76] A. Castañeda Miranda and V. Castañeda Miranda, "IoT-based fuzzy controller for hydroponic nutrient solution management in greenhouse crops," *Computers and Electronics in Agriculture*, vol. 176, p. 105638, 2020.

- [77] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, pp. 1877–1901, 2020.
- [78] J. Wei, M. Bosma, V. Y. Zhao, K. Guu, A. W. Yu, B. Lester, N. Du, A. M. Dai, and Q. V. Le, "Finetuned language models are zero-shot learners," in *International Conference on Learning Representations (ICLR)*, 2022.
- [79] Ollama, "Ollama: Get up and running with large language models locally." Software framework, available at <https://ollama.com>, 2023. Accessed: 2025.