PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA

MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH

YAHIA FARES UNIVERSITY OF MEDEA

Faculty of Science

Department of Mathematics and Computer Science



Graduation Project

Presented for obtaining the Master's Degree in Computer Science

Option: Applied Artificial Intelligence

THEME

Mobile Application for Plant Disease Detection Based On Combination Of Deep Learning And Image Segmentation

Prepared by:
ZEROUAL Rafik
TALBI Anis

Supervisor:
Mr. TOULAIT Soufiane

Jury:
CHETOUANE Mohamed
KHELDOUN Ahmed

Class of 2024/2025

الإهداء

❀

إلى أولئك الذين رسموا أحلامهم على جدران الجامعات،
وحملوا دفاترهم في طريقهم إلى الشهادة،
إلى طلبة **غزة** الجامعيين الذين رحلوا قبل أن تكتمل الحكاية،
قبل أن يُنادى بأسمائهم يوم التخرج،
فصاروا نورًا لا ينطفئ في سماء العلم والكرامة،
سلامًا على أرواحكم الطاهرة.

إلى زملائنا وأصدقائنا، الذين شاركونا هذه المرحلة بكل ما فيها من تحديات،
كنتم جزءًا من تجربة تعلمنا فيها الكثير معًا.

إلى جامعتنا، التي كانت فضاءً للدراسة والتجربة والنمو الأكاديمي،
نقدّر ما وفّرته لنا من فرص لبناء معارفنا ومهاراتنا.

وإلى أساتذتنا، الذين رافقونا في هذه الرحلة التعليمية،
نشكر كل من قدّم لنا المعلومة، وفتح لنا بابًا للفهم والتفكير.

نهدي هذا العمل إلى كل من كان جزءًا من هذه التجربة.

❀

Dedication

All praise and thanks are due to Allah, the Most Merciful, who gave us the strength, patience, and guidance to complete this work.

To **my dear parents**, whose unwavering support and endless love were the pillars of our journey.

To **my families**, for the encouragement, prayers, and faith they always carried for us.

To **my friends**, who walked beside us through every step — your presence, laughter, and sincere words made this journey lighter and more meaningful.

To **my teachers**, whose knowledge and guidance inspired us and helped us grow.

And to **my teammate and friend**, **Anis Talbi**, for being a true partner in this project, sharing every challenge and every moment of success.

Rafik Zeroual ❀

Dedication

First and foremost, I thank Allah for granting me the strength, patience, and guidance to complete this journey. Without His blessings, none of this would have been possible.

To **my parents**, I dedicate this work to you with deep love and gratitude. Thank you for your endless support, your sacrifices, and your constant prayers that have guided me throughout this journey.

To **my second mother, Zineb**, Thank you for your kindness, wisdom, and comforting presence.

To **my sister Zola**, her husband **Youcef**, and their children **Jod and Ranim**, Thank you for your love, your encouragement, and all the joyful moments we've shared.

To **my sisters Romissa and Khaoula**, Thank you for your care, your support, and your presence in my life.

To **my brothers Mohamed and Oussama** (and his family), Thank you for always believing in me, for your advice, and for being there when I needed you most.

To **my partner Rafik**, Thank you for your unwavering support, patience, and encouragement. Your presence has been a source of strength and inspiration throughout this project.

To **Kaouthar**, the friend of my life, Thank you for being by my side through everything. Your friendship means the world to me.

To **my dear friends Khaled and Mounir**, Thank you for your support, kindness, and motivation throughout this journey.

Anis Talbi ❀

Abstract

It's important for sustainable agricultural practice to accurately detect plant disease in order to facilitate early intervention to limit crop loss. This project describes a deep learning–based pipeline of image segmentation and classification in order to improve the reliability of plant disease detection. The segmentation process sought to separate the plant from its background to limit noise and allow the classification model to focus on features that were relevant. We experimented with many segmentation models and settled on U-Net with EfficientNet and Inception backbones because U-Net produced clean and accurate leaf masks. We evaluated many pretrained CNN architecture methods in the classification stage, and identified MobileNetV2 as a model that embodied the balance between speed, lightweight, and an impressive level of accuracy with the segmented leaf images. We will demonstrate that when deep learning segmentation and classification is combined with classification models, the performance of disease identification improves significantly. This project indicates the use of deep learning for agriculture applications works and promotes the development of smart, automated plant disease monitoring systems.

Keywords: Plant disease detection, Image segmentation, Classification,Deep learning, Transfer learning, Agricultural AI, Computer vision, Mobile Application,CNN .

Résumé

Pour une pratique agricole durable, il est essentiel de détecter avec précision les maladies des plantes afin de faciliter une intervention précoce et de limiter les pertes de récoltes. Ce projet décrit un pipeline de segmentation et de classification d'images basé sur l'apprentissage profond (Deep Learning) visant à améliorer la fiabilité de la détection des maladies des plantes. Le processus de segmentation visait à séparer la plante de son environnement afin de limiter le bruit et de permettre au modèle de classification de se concentrer sur les caractéristiques pertinentes. Nous avons expérimenté de nombreux modèles de segmentation et avons opté pour U-Net avec les backbones EfficientNet et Inception, car U-Net produisait des masques foliaires nets et précis. Nous avons évalué de nombreuses méthodes d'architecture CNN pré-entraînées lors de la phase de classification et avons identifié MobileNetV2 comme un modèle offrant un équilibre parfait entre rapidité, légèreté et une précision impressionnante pour les images foliaires segmentées. Nous démontrerons que l'association de la segmentation et de la classification par apprentissage profond aux modèles de classification améliore considérablement les performances d'identification des maladies. Ce projet démontre l'efficacité de l'apprentissage profond pour les applications agricoles et favorise le développement de systèmes intelligents et automatisés de surveillance des maladies des plantes.

Mots-clés : Détection de maladies des plantes, Segmentation d'images, Classification, Apprentissage profond, Apprentissage par transfert, IA agricole, Vision par ordinateur, Application mobile, CNN.

الملخص

من المهم للممارسات الزراعية المستدامة الكشف الدقيق عن أمراض النباتات لتسهيل التدخل المبكر للحد من خسائر المحاصيل. يصف هذا المشروع مسارًا قائمًا على التعلم العميق لتجزئة الصور وتصنيفها، بهدف تحسين موثوقية الكشف عن أمراض النبات. سعت عملية التجزئة إلى فصل النبات عن محيطه للحد من التشويش، وتمكين نموذج التصنيف من التركيز على السمات ذات الصلة. جربنا العديد من نماذج التجزئة، واستقرينا على U-Net مع شبكتي EfficientNet وInception، لأن U-Net أنتجت أقنعة أوراق دقيقة وواضحة. قيّمنا العديد من أساليب بنية CNN المُدربة مسبقًا في مرحلة التصنيف، وحددنا MobileNetV2 كنموذج يُجسد التوازن بين السرعة وخفة الوزن ومستوى الدقة المذهل في صور الأوراق المُجزأة. سنُثبت أنه عند دمج تجزئة وتصنيف التعلم العميق مع نماذج التصنيف، يتحسن أداء تحديد الأمراض بشكل ملحوظ. يُشير هذا المشروع إلى فعالية استخدام التعلم العميق في التطبيقات الزراعية، ويعزز تطوير أنظمة ذكية وآلية لمراقبة أمراض النبات.

الكلمات المفتاحية: كشف أمراض النبات، تجزئة الصور، التصنيف، التعلم العميق، التعلم الانتقالي، الذكاء الاصطناعي الزراعي، الرؤية الحاسوبية، تطبيقات الهاتف المحمول، شبكة .CNN

Contents

List of Figures **12**

List of Tables **13**

List of Abbreviations **14**

General Introduction **16**

Literature Review **21**

1.1 Introduction . . . 21
1.2 Machine Learning . . . 21
1.3 Deep Learning . . . 21
1.4 Deep Learning vs Machine Learning . . . 22
1.5 Supervised Learning . . . 22
1.6 Unsupervised Learning . . . 22
1.7 Artificial Neural Networks . . . 22
1.8 Deep learning concepts . . . 23
1.8.1 Forward Propagation . . . 23
1.8.2 Backpropagation . . . 24
1.8.3 Gradient Descent . . . 24
1.8.4 Epoch . . . 25
1.8.5 Batch Size . . . 25
1.8.6 Training Phase . . . 25
1.8.7 Validation Phase . . . 26
1.8.8 Testing Phase . . . 26
1.8.9 Overfitting . . . 26
1.8.10 Underfitting . . . 27
1.8.11 Loss Functions . . . 28
1.9 Recurrent Neural Networks (RNNs) . . . 29
1.10 Applications of deep learning . . . 29
1.11 Conclusion . . . 31

Deep Learning for Image Segmentation and Classification **34**

2.1 Introduction . . . 34
2.2 Image Segmentation Techniques . . . 35
2.2.1 Definition and Role in Plant Disease Detection . . . 35

2.2.2 Common Segmentation Techniques . . . 36
2.2.2.1 Traditional Methods . . . 36
2.2.2.2 Modern Deep Learning-Based Methods . . . 37
2.2.2.3 Comparative Analysis . . . 39
2.2.3 Segmentation Challenges in Agricultural Images . . . 40
2.3 Image Classification . . . 40
2.3.1 Role of Classification in Diagnosis . . . 40
2.3.2 Convolutional Neural Networks in Plant Disease Detection . . . 41
2.3.3 Transfer Learning in Plant Classification . . . 43
2.4 Related Work . . . 44
2.4.1 Image Segmentation . . . 44
2.4.2 Image Classification . . . 48
2.5 Conclusion . . . 49

Proposed System Architecture and Methodology **52**
3.1 Introduction . . . 52
3.2 Dataset Description and Preprocessing . . . 52
3.2.1 Dataset Overview . . . 52
3.2.1.1 Classification Dataset Overview . . . 52
3.2.1.2 Segmentation Dataset Overview . . . 53
3.2.2 Preprocessing Techniques . . . 55
3.2.3 Dataset Splitting . . . 56
3.3 Model Selection Criteria . . . 56
3.4 Segmentation Models Description . . . 57
3.4.1 Overview of Segmentation Approach . . . 57
3.4.2 Model Architectures . . . 57
3.4.3 Preprocessing and Input for Segmentation . . . 58
3.4.4 Ensemble Strategy . . . 58
3.5 Classification Models Description . . . 59
3.5.1 MobileNetV2 . . . 59
3.5.2 EfficientNetB3 . . . 60
3.5.3 VGG16 . . . 60
3.6 Project Requirements . . . 61
3.6.1 Development Environment . . . 61
3.6.2 Programming Languages . . . 61
3.6.3 Deep Learning and Machine Learning Libraries . . . 62
3.7 Evaluation Metrics . . . 63

3.7.1 Segmentation Metrics . . . 64
3.7.2 Classification Metrics . . . 64
3.8 Conclusion . . . 65

Model Evaluation and Application Deployment **68**
4.1 Introduction . . . 68
4.2 Model Evaluation and Discussion . . . 68
4.2.1 Segmentation Model Results and Analysis: . . . 69
4.2.1.1 Quantitative Results . . . 69
4.2.1.2 Comparative Analysis . . . 71
4.2.2 Classification Model Results and Analysis . . . 72
4.2.2.1 Training Configuration . . . 72
4.2.2.2 MobileNetV2 Results . . . 73
4.2.2.3 EfficientNetB3 Results . . . 75
4.2.2.4 VGG16 Results: . . . 77
4.2.3 Final Model Selection and Justification : . . . 79
4.2.3.1 Segmentation model . . . 79
4.2.3.2 Classification Model . . . 81
4.3 Development Environment . . . 83
4.4 Deployment in Mobile Application . . . 84
4.4.1 TFLite Conversion . . . 84
4.4.1.1 Classification Model . . . 84
4.4.1.2 Segmentation Model . . . 84
4.4.2 Flutter and Dart Integration . . . 84
4.4.3 Real-Time Inference Pipeline . . . 85
4.4.4 UI Features and User Experience . . . 86
4.5 Conclusion . . . 90

General Conclusion **92**

List of Figures

1.1 Artificial Neural Networks architecture . . . 23
1.2 Mechanism of forward propagation . . . 24
1.3 Mechanism of back propagation . . . 24
1.4 Overfitting scenario . . . 27
1.5 Underfitting scenario . . . 28
1.6 RNN architecture . . . 29
2.1 Example of segmented plant leaf images. . . . 36
2.2 U-Net architecture. . . . 38
2.3 Mask R-CNN architecture. . . . 38
2.4 DeepLab architecture. . . . 39
2.5 Convolution Layer Example with 32x32x1 Image and 5x5x1 Filter . . . 41
2.6 Two Pooling Methods Compared . . . 42
2.7 Comparing Neural Networks: With and Without Dropout . . . 42
2.8 How a Dense Layer Connects Neurons . . . 43
2.9 Schematic representation of CNN architecture applied to leaf disease detection . . . 43
3.1 Visual examples of plant diseases analyzed in this study. . . . 53
3.2 Visual examples of Segmentation Dataset . . . 54
3.3 Preprocessing and Augmentation Parameters . . . 55
3.4 Computing Class Weights to Handle Imbalanced Dataset . . . 55
3.5 Python Script for Splitting Dataset into Train, Test, and Validation Sets . . . 56
3.6 architecture of MobileNetV2 . . . 60
3.7 EfficientNetB3 architecture . . . 60
3.8 VGG16 architecture . . . 61
4.1 F1 Score comparison for all models on train, validation, and test sets. . . . 69
4.2 IoU Score comparison for all models on train, validation, and test sets. . . . 70
4.3 Loss comparison for all models on train, validation, and test sets. . . . 71
4.4 Training and Validation Accuracy and Loss of MobileNetV2 without Segmentation 73
4.5 Model Metrics Overview on Test Set Without Segmentation . . . 74
4.6 Training and Validation Accuracy and Loss of MobileNetV2 with Segmentation) . . . 74
4.7 Model Metrics Overview on Test Set With Segmentation) . . . 75
4.8 Training and Validation Accuracy and Loss of EfficientNetB3 without Segmentation) 75
4.9 Model Metrics Overview on Test Set Without Segmentation . . . 76
4.10 Training and Validation Accuracy and Loss of EfficientNetB3 with Segmentation . . . 76
4.11 Model Metrics Overview on Test Set With Segmentation . . . 77

4.12 Training and Validation Accuracy of VGG16 without Segmentation 78
4.13 Model Metrics Overview on Test Set Without Segmentation 78
4.14 Training and Validation Accuracy and Loss of VGG16 with Segmentation 79
4.15 Model Metrics Overview on Test Set With Segmentation 79
4.16 Segmentation result before and after applying Majority Voting. 81
4.17 Comparison of classification performance with and without segmentation. 82
4.18 Google Colab Logo 83
4.19 Visual Studio Code Logo 83
4.20 Real-Time Inference Pipeline for Plant Disease Detection Mobile App 86
4.21 Splash screen 87
4.22 Authentication Screens: Login and Sign-Up 87
4.23 Home page screens 88
4.24 Disease Detection Screens 88
4.25 Admin screen 89
4.26 profil screen 89

List of Tables

1.1 Comparison between Machine Learning and Deep Learning 22
2.1 Comparison between Traditional and Modern Deep Learning-Based Image Segmentation Methods 39
2.2 Overview of Image Segmentation Techniques in Plant Disease Detection 47
2.3 Comparison of Plant Disease Detection Architectures 50
3.1 Classification Dataset Summary 52
3.2 Segmentation Dataset Summary 54
4.1 Training Hyperparameters Used Across All Classification Models 72
4.2 Comparison of Ensemble Methods 80
4.3 Performance Metrics of Classification Models 82

List of Abbreviations

Abbreviation	Full Form
AI	Artificial Intelligence
ML	Machine Learning
DL	Deep Learning
ANN	Artificial Neural Network
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
ReLU	Rectified Linear Unit
Sigmoid	Sigmoid Activation Function
Tanh	Hyperbolic Tangent Activation Function
SGD	Stochastic Gradient Descent
MSE	Mean Squared Error
MAE	Mean Absolute Error
TFLite	TensorFlow Lite
IoU	Intersection over Union
VGG16	Visual Geometry Group 16
EfficientNetB3	EfficientNet version B3
MobileNetV2	Mobile Network Version 2
UI	User Interface
CPU	Central Processing Unit
GPU	Graphics Processing Unit

General Introduction

Agriculture continues to be fundamental to global food security and economic development and, still, is continually challenged to improve productivity and sustainability. The rise of Artificial Intelligence (AI) is providing new opportunities for agriculture to shift from static conventional practices to data-driven informed practices. This project examines crop health via AI through applications of Deep Learning and Computer Vision for plant disease detection. Efforts in agriculture involving health issues of the crop are critical to successful crop health management. The use of mobile technology and advanced image analysis will provide a new, innovative, accessible and scalable approach to support farmers with crops that are healthy and improving agricultural outcomes.

Context of the Study

In the past few years, Artificial Intelligence (AI) has progressed sharply and become the foundation of technological advancement. Various algorithms in AI — Machine Learning (ML), Deep Learning (DL), and Computer Vision — have applications in several fields including health care, self-driving cars, finance, and agriculture. Deep Learning has resulted in spectacular results in image and speech recognition, natural language processing, and even data analysis.

Agriculture is among the most crucial global industries that focuses on providing food and securing economic sustainability. Agribusiness faces many of the same challenges as several industries, including climate change, soil degradation, water supply threats, and disease impact on crops and livestock. AI's utilization in agriculture — known as precision agriculture — seeks to address these problems through automation by refining the processes of irrigation, diagnosing, and monitoring crop resilience and disease.

Various AI assignment techniques can now monitor crop health status using image-based systems. Through the function of Computer Vision capabilities, a machine can analyse leaf patterns and colour divergence to articulate and identify anomalies caused by pathogens. Further, many of these systems can be executed at scale in both large rotational farms and small conservation methods around the world, accessible via mobile applications. The convergence of AI, deep learning, and mobile technology opens new avenues to transform traditional farming into smart farming.

Problem Statement

The ability to promptly and accurately detect plant diseases is one of agriculture's most serious problems. In many cases, farmers make judgments based on visual inspection, a process that requires expertise and is often not entirely accurate, especially in the early stages of infection. This issue is even worse for farmers without access to agricultural experts, especially in rural and poorly developed areas. Diseases are often diagnosed only after damage has occurred, resulting in increased pesticide use, lower yield, and poorer product quality.

Plant disease identification faces challenges due to environmental variability: the same disease may appear differently under varied lighting, camera angles, and backgrounds. Moreover, traditional computer vision methods such as edge detection or colour filters struggle with such variability.

Consequently, there is a pressing need for intelligent systems that can autonomously identify plant diseases from leaf images despite these variations. Such systems can empower farmers with accurate and timely diagnoses, thus contributing to food security and sustainable agricultural practices.

Contribution of Artificial Intelligence and Deep Learning

Artificial Intelligence, particularly Deep Learning, has advanced the field of image recognition. In image classification, Convolutional Neural Networks (CNNs) are the standard due to their ability to learn hierarchical features from raw image data. CNNs are capable of recognizing complex features such as patterns in colour, spot shapes, and textures associated with plant diseases.

Transfer Learning further enhances performance by allowing the use of pre-trained models such as VGGNet, ResNet, MobileNet, and EfficientNet. These models, trained on millions of images, can be adapted to specific tasks like plant disease classification, even with small datasets, thereby saving time and improving accuracy.

Image Segmentation is also a valuable method, as it enables the separation of the leaf from the background. This helps reduce background noise and allows the model to focus solely on the relevant plant features, thus improving classification performance.

Together, CNNs, Transfer Learning, and Segmentation techniques form a powerful toolkit for accurate and scalable plant disease detection.

Mobile Application Role

The widespread availability of smartphones, even in rural areas, makes mobile applications a practical medium for deploying AI systems. With a mobile app, users can take a photo of a leaf and receive an instant diagnosis regarding the plant's health.

From a technical standpoint, deploying deep learning models on mobile devices requires reducing model size while maintaining accuracy. Frameworks such as TensorFlow Lite and ONNX enable the conversion and compression of models for efficient mobile deployment. The mobile application must also have a user-friendly interface to ensure accessibility for users with minimal technical experience.

Benefits of the mobile app include:

- Empowering farmers with decision-making tools.

- Reducing dependency on external experts or labs.
- Enabling efficient monitoring of large fields.
- Allowing early detection and timely intervention.

This integration of AI and mobile technology results in a smart, accessible, scalable, and impactful solution.

5. Motivation and Objectives

The motivation behind this project stems from the desire to apply our academic knowledge of Artificial Intelligence to solve real-world problems. Agriculture, being a critical domain for any country's economy and food security, presents both a challenge and an opportunity for AI applications.

As AI engineering students, we are motivated by the potential to create something that combines theory with practice — not just to simulate models on academic datasets, but to deploy them in practical, life-changing ways. This project allows us to explore all major components of our field of study: model training, computer vision, mobile development, and deployment.

The main objectives of our project are as follows:

- To build a deep learning model capable of detecting and classifying plant diseases from leaf images.
- To enhance model performance through image segmentation techniques that isolate the leaf from the background.
- To integrate the model into a functional mobile application that is user-friendly and accessible to farmers and agricultural engineers.

Organization of Work

This work consists of four chapters beside a general introduction and general conclusion. A description of each chapter is presented in the following lines :

- **Chapter 1 :**
  This chapter introduces the basic concepts of machine learning and deep learning, focusing on the methods relevant to image classification. It explains key ideas behind neural networks and training processes, and highlights the importance of deep learning for tasks like plant disease detection.

- **Chapter 2 :**
  This chapter reviews key image segmentation and classification techniques relevant to plant disease detection. It covers traditional and deep learning–based segmentation methods, the challenges specific to agricultural images, and the role of classification in diagnosing plant diseases. The chapter also discusses the use of Convolutional Neural Networks and transfer learning for improving classification accuracy. Finally, it summarizes related work in both segmentation and classification fields.

- **Chapter 3 :**
  This chapter presents the models and system architecture used in the project. It starts by describing the classification and segmentation datasets, including preprocessing steps and data splitting. Then, it explains how models were selected and details the segmentation models used, including their architecture and input handling. For classification, the chapter introduces MobileNetV2, EfficientNetB3, and VGG16, along with training configurations. It also outlines the project's software environment, programming tools, and machine learning libraries. Finally, it defines evaluation metrics for both segmentation and classification and concludes with a transition to the next stage of the project.

- **Chapter 4 :**
  This chapter focuses on evaluating the performance of the proposed models and preparing them for deployment. It will begin by presenting the results of both segmentation and classification tasks using the evaluation metrics defined earlier. Performance comparisons between different models will be discussed. The chapter will then demonstrate visual results to show how well the models perform on real data. Finally, it will describe the deployment process, including how the selected models are integrated into a user-friendly mobile application for practical plant disease detection in agricultural settings.

Chapter 1

Literature Review

1.1 Introduction

The world is witnessing rapid technological development, and Artificial Intelligence (AI) stands out as one of the most transformative innovations across various sectors—from healthcare and agriculture to industry and education. AI refers to the ability of computer systems to simulate human capabilities such as understanding, learning, decision-making, and problem-solving.

Through techniques like Machine Learning and Deep Learning, computers can now analyze vast amounts of data, identify patterns, and make accurate decisions without explicit programming. This progress has made AI a powerful tool for driving innovation and simplifying everyday life.

In this context, this study aims to explore the fundamental concepts of AI, with a focus on how these technologies can be harnessed in practical applications such as early plant disease detection—contributing to smart agriculture and enhancing food security.

1.2 Machine Learning

Machine learning involves creating models by training an algorithm to make predictions or decisions based on data [1]. It encompasses techniques including:

- Linear and logistic regression
- Decision trees and random forests
- Support vector machines (SVMs)
- k-nearest neighbors (KNN)
- Clustering algorithms

1.3 Deep Learning

Deep learning is a subset of machine learning, which is itself a branch of artificial intelligence (AI). It is called "deep" because it uses deep neural networks with multiple layers to process data and make decisions. Deep learning enables computers to learn from experience and understand the world through a hierarchy of concepts. These layers allow the machine to build complex ideas from simpler ones, much like humans do when analyzing data. If visualized, this hierarchy forms a deep structure with many layers, giving deep learning its name [2].

1.4 Deep Learning vs Machine Learning

Machine learning and deep learning are both branches of artificial intelligence (AI). Machine learning enables systems to adapt and improve automatically with minimal human intervention. Deep learning, a subset of machine learning, takes this further by using artificial neural networks (ANNs) to mimic the learning process of the human brain [3].

Machine Learning	**Deep Learning**
A subset of AI	A subset of machine learning
Works with smaller datasets	Requires large amounts of data
Needs human intervention for corrections	Learns autonomously from experience
Trains faster but may have lower accuracy	Takes longer to train but achieves higher accuracy
Identifies simple, linear patterns	Detects complex, non-linear relationships
Can train on a CPU	Requires a GPU for efficient training

Table 1.1: Comparison between Machine Learning and Deep Learning [3]

1.5 Supervised Learning

Supervised learning is a machine learning technique in which an algorithm learns to map inputs to outputs using labeled example pairs. The objective is to accurately approximate this mapping so that when new input data is provided, the algorithm can predict the corresponding output [4].

1.6 Unsupervised Learning

Unsupervised Learning refers to a set of machine learning techniques that aim to discover underlying structures or distributions in input data without the use of labeled examples. The primary goal is to model the underlying structure or distribution in the data to learn more about it [4].

1.7 Artificial Neural Networks

Artificial Neural Networks (ANNs), or simply neural networks, are computational models inspired by the structure and function of the human brain. They consist of **an input layer, one or more hidden layers, and an output layer,** all composed of interconnected neurons that process and transmit information. **The input layer** receives raw data,**the hidden layers** apply transformations through

weighted connections, and **the output layer** generates the final prediction or decision. These networks learn patterns from data through training, making them fundamental in machine learning and artificial intelligence applications [5].

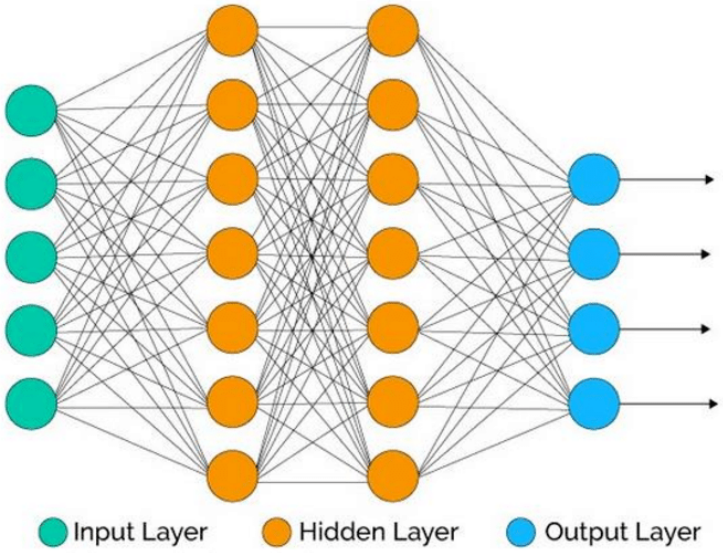


Figure 1.1: Artificial Neural Networks architecture

1.8 Deep learning concepts

1.8.1 Forward Propagation

Forward propagation is a fundamental process in neural networks, especially within the field of artificial intelligence (AI). It involves the computation and transmission of input data through the network layers to produce an output. The data flows in a forward direction: from the input layer, through the hidden layers, and finally to the output layer. During this process, the data does not loop back at any point [6].

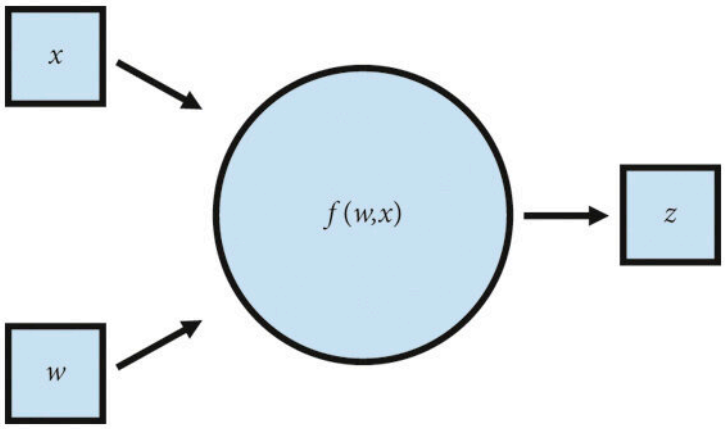


Figure 1.2: Mechanism of forward propagation

1.8.2 Backpropagation

Backpropagation is a crucial process in training neural networks. It works by taking the error generated during forward propagation and transmitting it backward through the network. This allows the model to adjust the weights of its connections, improving accuracy over multiple iterations. By minimizing the error, backpropagation helps the neural network learn more effectively. [6].

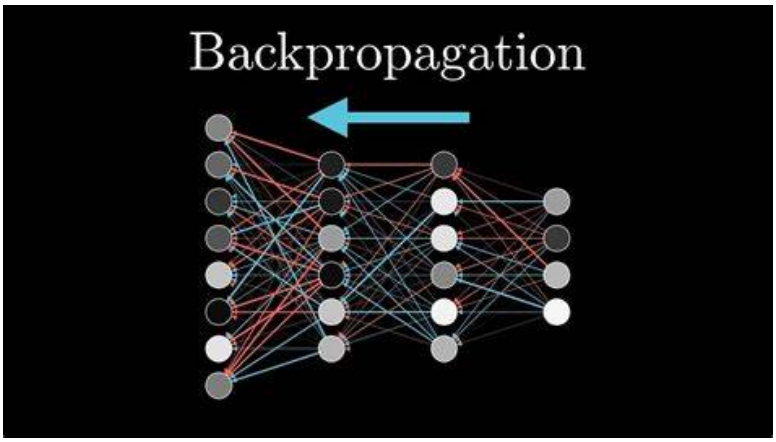


Figure 1.3: Mechanism of back propagation

1.8.3 Gradient Descent

Gradient descent is a core optimization method used in various machine learning algorithms, such as linear regression, logistic regression, support vector machines, and neural networks. It helps minimize the model's cost function by gradually adjusting the model's parameters, reducing the discrepancy between predicted and actual outcomes, which ultimately enhances the model's accuracy and performance [7].

1.8.4 Epoch

an epoch is defined as one complete pass through the entire training dataset. In practical terms, this means that during one epoch, every example in the dataset is used once to update the model's parameters. Training typically involves multiple epochs, allowing the model to iteratively refine its weights and improve its performance [8].

1.8.5 Batch Size

Batch size is a critical hyperparameter in deep learning that determines the number of training samples processed before the model's internal parameters are updated. It plays a significant role in the training process, affecting both the speed and stability of learning.

- There are three main types based on batch size:
  - **Batch Gradient Descent**: Uses the entire dataset as one batch.
  - **Stochastic Gradient Descent (SGD)**: Uses one sample per batch.
  - **Mini-Batch Gradient Descent**: Uses a subset of samples (e.g., 32, 64, 128).
- Larger batch sizes require more memory, provide stable gradient updates, but may converge slower.
- Smaller batch sizes use less memory, introduce noise that can aid generalization, and converge faster but with more oscillations.
- In practice, batch sizes are often set to powers of 2 (e.g., 32, 64, 128) for computational efficiency [9]

1.8.6 Training Phase

The training phase is the core process in machine learning where the model learns from labeled data. During training, the model adjusts its internal parameters (weights and biases) to minimize the loss function, which measures the difference between the model's predictions and the actual labels. The training dataset is typically the largest portion of the data, often around 60-80% of the total dataset. This phase is iterative, with the model improving its performance over multiple epochs (passes through the data). The goal is to ensure the model captures the underlying patterns in the data without memorizing it, which could lead to overfitting. [9].

1.8.7 Validation Phase

The validation phase is used to evaluate the model's performance during training and to fine-tune hyperparameters, such as learning rate, batch size, and network architecture. The validation dataset is separate from the training data and is not used for learning. It helps detect overfitting, where the model performs well on the training data but poorly on unseen data. Validation metrics, such as accuracy or loss, guide decisions about when to stop training or adjust hyperparameters. Typically, the validation set constitutes 10-20% of the total dataset. Techniques like cross-validation can be used when data is limited to ensure robust evaluation. [9].

1.8.8 Testing Phase

The testing phase provides the final evaluation of the model's performance on completely unseen data. The test dataset is used only after the model is fully trained and validated. It simulates real-world scenarios, offering an unbiased estimate of how the model will perform in practice. The test set is typically 10-20% of the total dataset and should never be used during training or validation. Testing metrics, such as accuracy, precision, recall, or F1 score, provide a comprehensive understanding of the model's generalization ability and readiness for deployment. [9].

1.8.9 Overfitting

When a machine learning model gets too good at learning the training data, it's called overfitting. This means the model not only picks up on the important patterns but also on the random noise and unimportant details. As a result, the model might do really well on the training data but struggle when faced with new, unseen data like validation or test sets. Overfitting is a common issue, especially with complex models like deep neural networks, because they can end up just memorizing the training data instead of actually learning from it.

Overfitting happens when the model is too complicated compared to the amount and quality of the training data. For example, if you have a deep neural network with lots of layers and neurons but only a small or noisy dataset, the model might just memorize everything instead of truly understanding it. Other causes of overfitting include training for too long, using irrelevant or redundant features, or having too many parameters. [9].

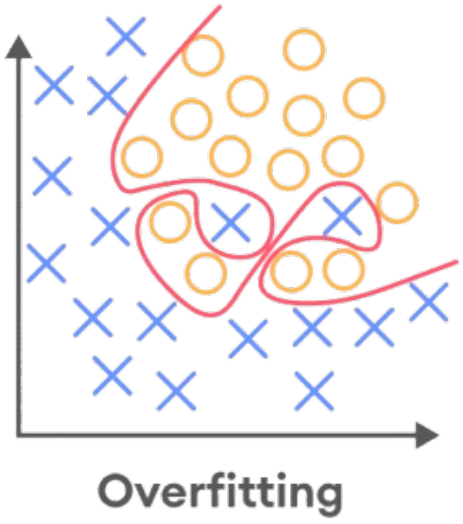


Figure 1.4: Overfitting scenario

1.8.10 Underfitting

Underfitting occurs when a machine learning model is too simple to capture the underlying patterns in the training data, resulting in poor performance on both the training and validation/test sets. It is often caused by insufficient model complexity, inadequate training, poor feature engineering, or excessive regularization. Signs of underfitting include high training and validation loss, low performance metrics, and learning curves that plateau at high error levels. To address underfitting, one can increase model complexity, improve feature selection, reduce regularization, train for longer, or use more advanced algorithms. [9].

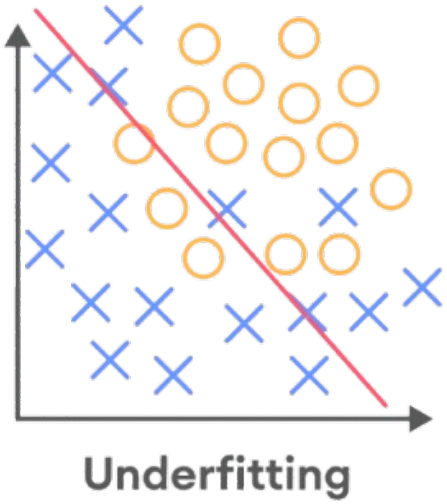


Figure 1.5: Underfitting scenario

1.8.11 Loss Functions

A loss function measures the difference between a model's predictions and the actual target values, guiding the optimization process during training. Common types include:

- **Mean Squared Error (MSE)**: Used for regression, calculates the average squared difference.
- **Mean Absolute Error (MAE)**: Also for regression, calculates the average absolute difference.
- **Cross-Entropy Loss**: Used for classification, measures the difference between predicted and true probability distributions.
- **Hinge Loss**: Used in SVMs for binary classification.
- **KL Divergence**: Measures the difference between two probability distributions, often used in probabilistic models.

The loss function plays a key role in training, evaluation, and regularization.[9]

1.9 Recurrent Neural Networks (RNNs)

Recurrent neural networks (RNNs) are a class of neural networks designed to process sequential data inputs and transform them into sequential outputs. They are particularly suited for handling data where sequence elements are interrelated, such as in natural language text, speech signals, or time-series measurements [10].

Unlike conventional feedforward networks, RNNs maintain an internal state that captures information about previous inputs in the sequence. This is achieved through recurrent connections where the output from the previous time step is fed back as input to the current computation [11]. The basic RNN structure can be represented as a single layer that is unfolded across time steps, with parameters A, B, and C governing the transformations between inputs, hidden states, and outputs respectively (Figure 1.13).

The recurrent architecture allows the network to exhibit dynamic temporal behavior and learn patterns in sequential data. However, basic RNNs often struggle with learning long-range dependencies due to the vanishing gradient problem during training. This limitation has led to the development of more advanced variants like Long Short-Term Memory (LSTM) networks and Gated Recurrent Units (GRUs), which incorporate gating mechanisms to better control information flow through the sequence [12].

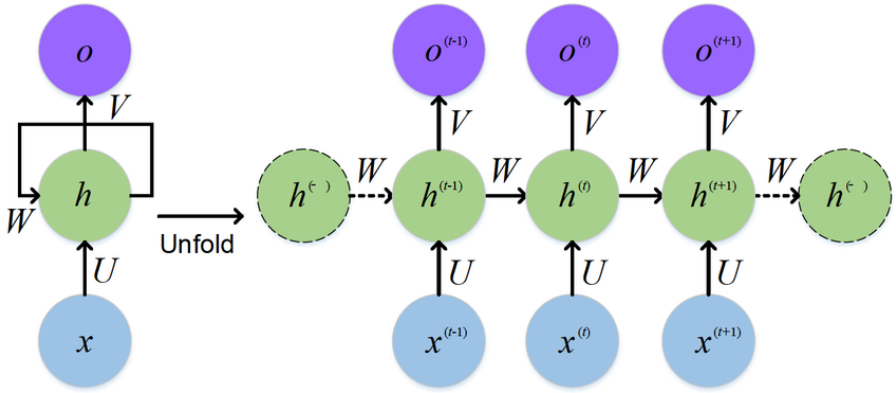


Figure 1.6: RNN architecture

1.10 Applications of deep learning

Deep learning is considered one of the breakthrough technologies in the field of artificial intelligence (AI), as it empowers systems with abilities to learn and extract the hierarchical features inherently within large and unstructured data. Contrary to classical machine learning models – for which extensive manual feature engineering is required –, deep learning strategies operate directly on raw data, successfully reaching state-of-the-art results in multiple settings. Its uses include computer vision, natural language processing, speech processing, healthcare, autonomous systems,

cyber security, agriculture, scientific research and more.[13].

Computer Vision Computer vision was the site of one of the first and most important advances in deep learning. Convolutional neural networks, or CNNs, are now the basis for tasks like generative modeling, object detection, image classification, and semantic segmentation [14, 15]. In image recognition tasks like ImageNet, models like AlexNet, VGG, ResNet, and DenseNet have shown exceptional performance. In order to improve diagnostic workflows and enable automated medical image analysis, U-Net and related CNN architectures are utilized in the medical field for precise organ, lesion, and tumor segmentation[16].

Natural Language Processing The advent of deep learning models that can comprehend and produce human language has revolutionized natural language processing. For sequential tasks like sentiment analysis, translation, and named entity recognition, Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM) networks, and Gated Recurrent Units (GRUs) have long been employed[17, 18]. Recent advances in question answering, summarization, and machine translation have been made possible by transformer-based architectures like T5, BERT (Bidirectional Encoder Representations from Transformers), and GPT (Generative Pre-trained Transformer). These models perform well on a variety of linguistic tasks because they make use of attention mechanisms and pretraining on extensive text corpora[19, 20, 21].

Speech and Audio Processing By enabling more precise and reliable systems for automatic speech recognition (ASR), speaker identification, and voice synthesis, deep learning has improved speech and audio processing. While architectures like WaveNet and Tacotron produce human-like speech in text-to-speech (TTS) systems[22, 23], deep acoustic models employing DNNs, LSTMs, and CNNs extract high-level features from unprocessed audio signals. These days, assistive technology, transcription services, and voice assistants all rely on these technologies[24].

Time-Series Forecasting Industries like manufacturing, transportation, and finance frequently use time-series data, which is defined by temporal dependencies. To predict future values, identify anomalies, and model intricate temporal patterns, deep learning models—in particular, LSTMs and 1D CNNs—are employed. They are used, for instance, in real-time IoT sensor data analysis, predictive maintenance in industrial settings, energy monitoring, and stock price prediction[25].

Healthcare and Biomedical Engineering Deep learning's capabilities have greatly benefited the healthcare industry. In order to identify conditions like cancer, pneumonia, and cardiovascular abnormalities, CNNs are used to analyze radiological images (such as X-rays, CT scans, and MRIs)[26]. RNNs and transformer models are used to mine Electronic Health Records (EHRs) in order to support clinical decision-making, suggest treatment plans, and predict patient outcomes[27]. Deep

generative models also aid in drug discovery by modeling chemical interactions and forecasting molecular structures [28].

Autonomous Systems and Robotics Deep learning is essential for perception, control, and decision-making in robotics and autonomous cars. CNNs manage lane detection, object recognition, and scene comprehension in self-driving cars, while reinforcement learning (RL) algorithms allow driving policies to be learned through trial-and-error interactions. Deep RL makes it easier to create control strategies for robotics tasks like grasping, moving, and planning a path in dynamic environments[29].

Cybersecurity Deep learning improves the detection of complex threats in the field of cybersecurity. RNNs and autoencoders simulate system behaviors and identify irregularities in user behavior or network traffic that could indicate fraud or intrusion. More proactive and adaptable security infrastructures are made possible by the use of CNNs and Graph Neural Networks (GNNs) for malware analysis, phishing attack detection, and real-time system vulnerability monitoring. [30, 31].

Agriculture Precision farming and sustainable crop management are two areas of modern agriculture where deep learning is becoming more and more important. By examining leaf images, CNNs can detect visual signs of infections, pests, and nutrient deficiencies, which is how plant diseases are detected[32]. Crop monitoring, yield prediction, and land use classification are supported by deep learning models that process drone and satellite imagery. Algorithms for object detection also aid in weed identification, fruit counting, and plant growth stage evaluation. Deep reinforcement learning is used in smart farming systems to optimize fertilization and irrigation schedules according to crop and environmental conditions. In addition to improving agricultural output, these technologies also lessen their negative effects on the environment and resource consumption[33]..

Scientific Research and Climate Modeling In domains like climate modeling, materials science, and genomics, deep learning is speeding up scientific research. As evidenced by AlphaFold's breakthrough in protein folding, models in bioinformatics can classify DNA sequences, predict gene functions, and infer protein structures[34]. Deep learning is used in environmental science for climate simulation, weather forecasting, and remote sensing data analysis to identify natural disasters, deforestation, and glacier retreat. These developments improve our comprehension of intricate natural systems and encourage data-driven research [35]..

1.11 Conclusion

By outlining the key ideas that set hierarchical, data-driven representation learning apart from more conventional methods, this chapter has followed the development of machine learning into

the deep learning era. We looked at important ideas like gradient-based optimization, forward and backward propagation, and the structural components of contemporary neural networks Innovative applications in computer vision, natural language processing, healthcare, autonomous systems, and other fields have been made possible by deep learning's extraordinary ability to extract noteworthy features from unprocessed data. However, issues like the requirement for sizable labeled datasets, vulnerability to hostile perturbations, and the interpretability of deep models continue to be research priorities.

These constraints should be lessened in the future by developments in self-supervised learning, model compression, and explainable AI, which will make deep networks more transparent in their decision-making, deployable on edge devices, and data-efficient. Deep learning is positioned to play a key role in the next generation of intelligent systems, driving both theoretical insights and practical innovations as computational resources become more widely available and algorithms become more complex.

Chapter 2

Deep Learning for Image Segmentation and Classification

2.1 Introduction

The integration of image analysis techniques into agricultural practices has revolutionized the way plant health is monitored and managed. With the advent of powerful computer vision and artificial intelligence (AI) tools, automated systems are increasingly being developed to assist in tasks such as crop monitoring, yield estimation, weed detection, and most notably, disease diagnosis [36]. These systems aim to reduce human error, lower operational costs, and enable timely interventions that can prevent the spread of diseases and minimize yield losses.

In the context of plant disease detection, digital image analysis involves processing visual data to identify symptoms of infections or stress on plant organs such as leaves, stems, or fruits. The core components of such analysis are **segmentation** and **classification**—two stages that are technically and conceptually distinct but deeply interconnected.

The Role of Segmentation in Image Analysis

Segmentation refers to the process of dividing an image into meaningful and non-overlapping regions, each corresponding to a specific object or area of interest. In agricultural applications, this often involves isolating plant parts—such as leaves—from complex and heterogeneous backgrounds that may include soil, mulch, weeds, or shadows [37]. These background elements can introduce significant noise into the data and reduce the performance of subsequent analysis tasks.

In our project, segmentation is employed primarily as a *preprocessing step* aimed at removing the background and isolating the plant leaf region. This is particularly important when working with real-world images captured in natural environments, where lighting conditions and background textures are highly variable. By segmenting out only the leaf area, we ensure that the model focuses exclusively on relevant visual features, such as color changes, texture variations, or lesion patterns, that are indicative of plant diseases [38].

For segmentation, we have employed advanced deep learning models such as **U-Net** and its variants, which are well-suited for pixel-wise classification tasks in biomedical and agricultural imaging [39]. These models allow for precise boundary detection, even in challenging conditions, and serve as a critical foundation for the next stage: classification.

The Role of Classification in Disease Diagnosis

Once the region of interest (i.e., the plant leaf) has been segmented, the next step is to determine whether it is healthy or affected by a specific disease. This process is referred to as **classification**, and it typically involves analyzing the visual features within the segmented region to assign it a disease label.

Classification is performed using machine learning or deep learning models trained on labeled

datasets, where each image is associated with a known disease category. Convolutional Neural Networks (CNNs), in particular, have demonstrated state-of-the-art performance in various image classification tasks due to their ability to learn hierarchical and spatially invariant features [40]. In our project, the classification step is built upon the output of the segmentation network, ensuring that only the relevant plant region is considered, thereby increasing the classification accuracy and reducing false positives.

Importance of Combining Segmentation and Classification

The combined use of segmentation and classification creates a robust and interpretable pipeline for automated plant disease detection. Segmentation ensures spatial precision by filtering out irrelevant parts of the image, while classification provides semantic interpretation by assigning meaningful labels to the segmented regions. When used together, these techniques reduce the noise, improve the focus of the model, and enable fine-grained analysis of disease symptoms.

Moreover, this combination allows for better scalability and adaptability of the detection system across different crops and environmental conditions. For example, the same segmentation network can be used across multiple datasets, while the classification module can be retrained or fine-tuned to recognize different diseases as needed.

2.2 Image Segmentation Techniques

2.2.1 Definition and Role in Plant Disease Detection

Image segmentation refers to the process of partitioning an image into multiple segments or regions, each of which corresponds to a different part or object within the scene. In the context of plant disease detection, the main objective of segmentation is to isolate the plant (especially its leaves) from the background (soil, sky, or other irrelevant parts). This isolation ensures that subsequent image analysis focuses exclusively on the plant regions of interest, improving accuracy in disease detection. By segmenting plant regions, we can:

- **Isolate infected areas:** This allows for more precise detection and classification of plant diseases.
- **Remove background noise** : Non-relevant parts of the image (soil, sky, etc.) are excluded, leading to cleaner and more accurate analyses.
- **Quantify disease severity:** The segmented infected regions can be measured to estimate the extent of infection.

- **Enable automated systems:** Segmentation prepares the image for machine learning models, making automated disease detection feasible and efficient.

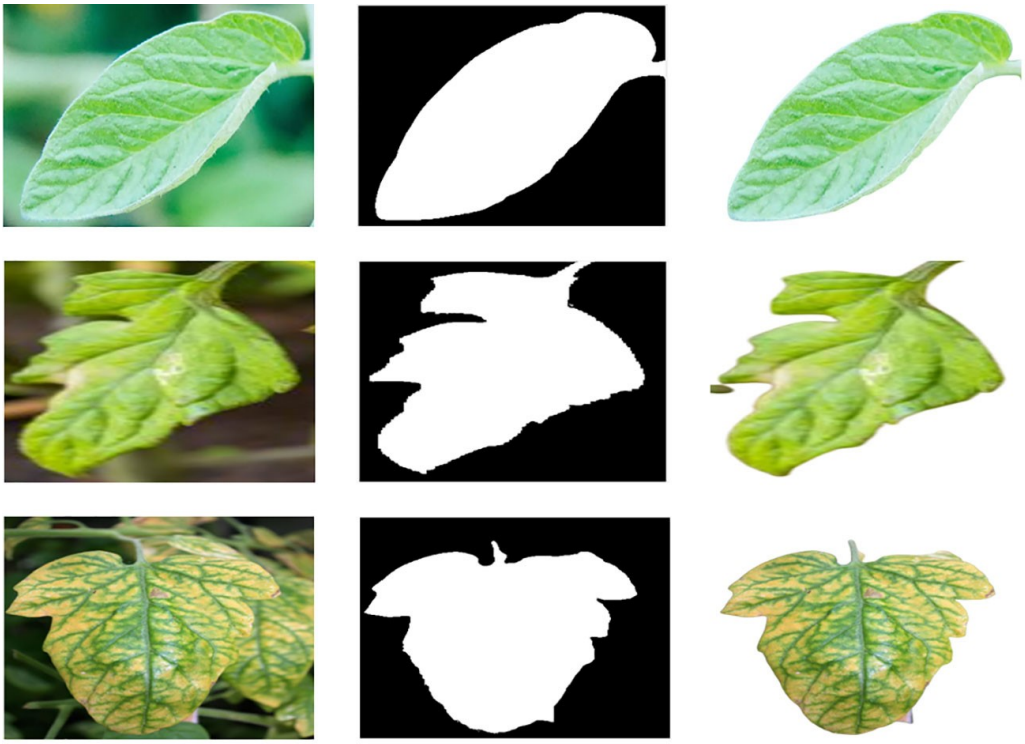


Figure 2.1: Example of segmented plant leaf images.

2.2.2 Common Segmentation Techniques

Image segmentation techniques can be broadly classified into two categories: traditional methods and modern deep learning-based methods.

2.2.2.1 Traditional Methods

Traditional image segmentation methods are classical computer vision techniques that operate directly on pixel intensities without requiring learning from data. These methods rely on predefined mathematical rules and heuristics to delineate object boundaries or homogeneous regions in an image. While they are computationally efficient and suitable for simple environments, they often struggle under varying conditions commonly observed in agricultural imagery.

- **Thresholding** : Thresholding is one of the simplest segmentation techniques. It involves classifying pixels into foreground and background based on their intensity values relative to a predefined threshold. A popular variant is Otsu's [41] method , which automatically determines the optimal threshold by minimizing intra-class intensity variance. In plant images, thresholding can be used to separate leaves from a uniformly colored background but is sensitive to lighting variations and complex backgrounds.

- **Edge Detection :**Edge detection techniques aim to identify significant changes in intensity, which often correspond to object boundaries. The Canny edge detector[42] is a widely used algorithm that detects edges by finding areas of rapid intensity change, followed by edge linking and suppression of false edges. Contour-based methods can delineate leaf margins but often produce fragmented or noisy boundaries in cluttered agricultural scenes.

- **Region-Based Methods :** Region-based methods segment an image by grouping together pixels that exhibit similar characteristics such as color, texture, or intensity. A notable technique is watershed segmentation [43], which treats the grayscale image as a topographic surface and identifies catchment basins corresponding to homogeneous regions. These methods are effective when object regions are distinct but may lead to over-segmentation when noise or gradients are present.

2.2.2.2 Modern Deep Learning-Based Methods

Modern deep learning-based segmentation techniques have significantly advanced the field by automatically learning hierarchical features from data, making them highly effective for complex and variable agricultural images. These models, primarily based on convolutional neural networks (CNNs), do not require manual feature engineering and can generalize well across different environments.

- **CNNs :** CNNs are a class of deep neural networks that automatically learn spatial hierarchies of features through convolutional layers. Introduced by LeCun et al. (1998), CNNs can extract low-level features (edges, textures) and high-level features (shapes, objects) from images, making them suitable for both classification and segmentation tasks. [44].

- **FCNs :** FCNs (Long, Shelhamer, Darrell, 2015) extend CNNs by replacing the fully connected layers with convolutional layers, enabling dense, pixel-wise classification. FCNs generate segmentation maps of the same spatial resolution as the input image, making them foundational for semantic segmentation tasks in agriculture[45].

- **U-Net :** U-Net (Ronneberger, Fischer, Brox, 2015) is a specialized architecture designed for biomedical and agricultural image segmentation. It employs a symmetric encoder-decoder structure with skip connections, which combine low-level spatial information with high-level semantic information. U-Net is particularly effective in scenarios with limited training data and has been widely adopted for leaf and disease segmentation tasks[39].

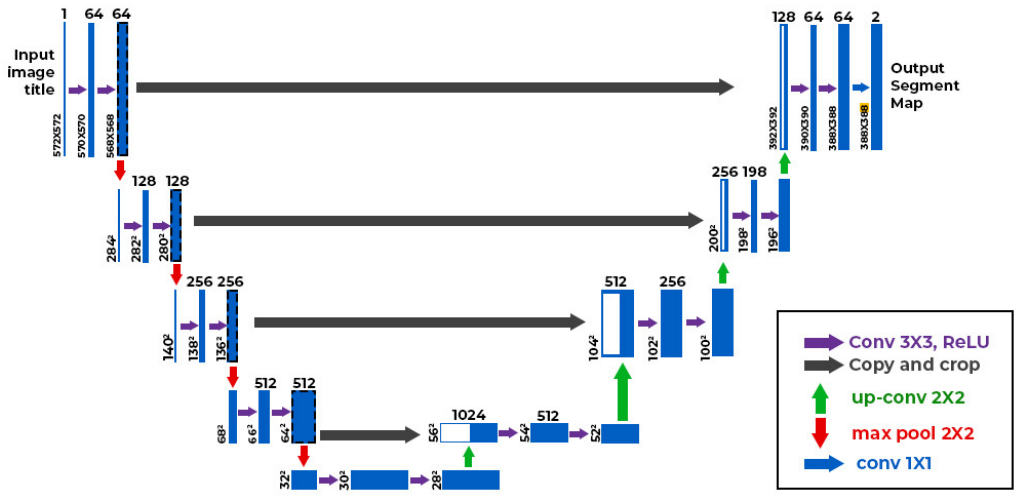


Figure 2.2: U-Net architecture.

- **Mask R-CNN :** Mask R-CNN (He et al., 2017) extends object detection frameworks by adding a branch that predicts binary masks for individual object instances, enabling instance segmentation. Unlike semantic segmentation (which classifies each pixel into a category), instance segmentation distinguishes between different instances of the same object class (e.g., multiple leaves). [46].

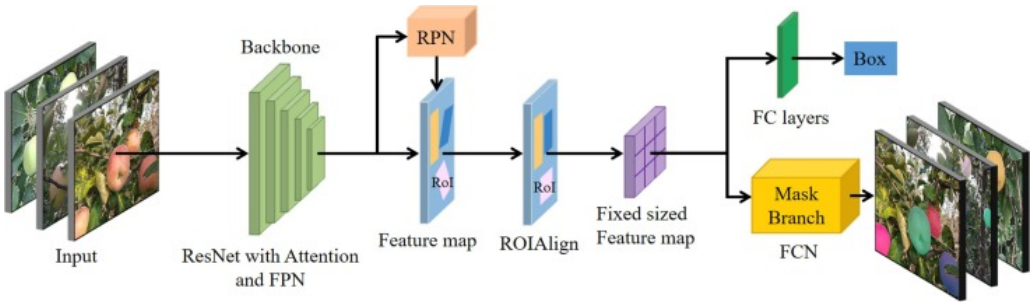


Figure 2.3: Mask R-CNN architecture.

- **DeepLab :** DeepLab (Chen et al., 2017) introduces advanced techniques such as atrous (dilated) convolutions and spatial pyramid pooling to capture multi-scale context, making it robust to scale variations. DeepLab has shown excellent performance in segmenting complex, high-resolution plant imagery where disease symptoms manifest at various scales. [47].

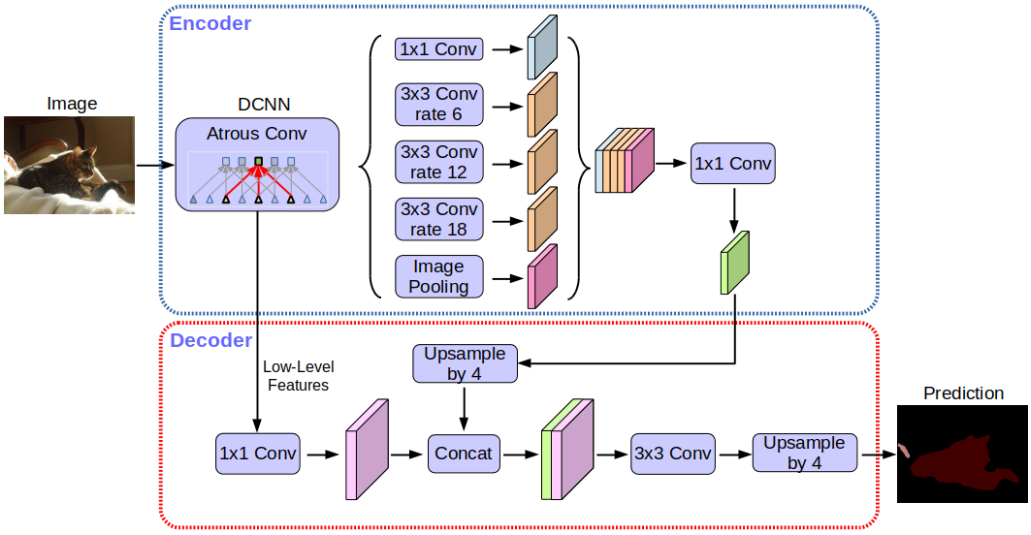


Figure 2.4: DeepLab architecture.

2.2.2.3 Comparative Analysis

The two categories of segmentation techniques—traditional and deep learning-based—differ significantly in methodology, performance, and applicability, especially in agricultural imaging scenarios. While traditional methods are lightweight and easier to implement, they often lack robustness and adaptability. In contrast, deep learning methods offer higher accuracy and generalization capabilities but require substantial data and computational resources. Table 2.1 provides a comparison between these two approaches across several important aspects.

Aspect	**Traditional Methods**	**Modern Deep Learning Methods**
Approach	Rule-based algorithms (e.g., thresholding, edges)	Data-driven feature learning (CNNs)
Data Requirement	No need for large datasets	Requires large annotated datasets
Computational Complexity	Low to moderate	High (requires GPUs)
Robustness to Noise	Low to moderate	High
Adaptability	Limited to specific conditions	Generalizes to complex, varied conditions
Accuracy	Moderate	High
Application Example	Otsu thresholding, Canny edge detection	U-Net, Mask R-CNN, DeepLab

Table 2.1: Comparison between Traditional and Modern Deep Learning-Based Image Segmentation Methods

2.2.3 Segmentation Challenges in Agricultural Images

Despite the advancements in image segmentation, several challenges arise in agricultural image processing:

- **Lighting variability** : Changes in sunlight, shadows, or reflections can affect color and contrast.
- **Background noise** : Soil, grass, or other plants can introduce noise that complicates segmentation.
- **Overlapping leaves** : Plant leaves often overlap, making it difficult to separate individual parts.
- **Variable disease symptoms** : Symptoms such as lesions or spots vary in shape, size, and color.
- **Occlusions and artifacts** : Presence of insects, dirt, or water droplets may affect segmentation quality.

Addressing these challenges is critical to developing robust, real-world plant disease detection systems .
Image segmentation is a key pre-processing step in plant disease detection. By accurately isolating the plant from the background, it enables precise detection of disease symptoms and facilitates the application of machine learning techniques for automated diagnosis. Both traditional methods and modern deep learning-based approaches offer valuable tools, with deep learning models providing enhanced performance in complex agricultural scenarios.

2.3 Image Classification

2.3.1 Role of Classification in Diagnosis

Image classification is a fundamental task in computer vision, aiming to assign predefined labels to images based on their visual content. In the context of plant disease detection, classification enables the automated identification of the type and severity of disease affecting a plant based on leaf imagery. This process turns visual symptoms—such as discoloration, texture change, or shape deformation—into meaningful disease categories. When integrated into precision agriculture systems, classification models support early intervention, limit crop damage, and enhance yield outcomes.

2.3.2 Convolutional Neural Networks in Plant Disease Detection

Convolutional Neural Networks (CNNs) have revolutionized image classification tasks, including plant disease detection. Unlike traditional approaches that rely on handcrafted features and manual tuning, CNNs automatically learn hierarchical representations of image features directly from the data. Their structure—designed to mimic the visual cortex—makes them highly effective in identifying complex patterns in leaf textures, colors, and shapes that correspond to various plant diseases.

A CNN typically consists of several key layers, each with a specific role in processing visual information:

Convolutional Layers A convolution layer is a base element of CNNs, which is mainly utilized to capture local patterns from input data, e.g., images. A small matrix, known as the filter (or kernel), is slid over the input, performing an element-wise multiplication and summing the result to get a new representation—the feature map. This allows the network to learn patterns such as edges and textures as well as shapes [48].

Key Components of a Convolutional Layer

- **Filters (or Kernels):** Small, trainable matrices applied to the input data to capture relevant patterns. Each filter is specialized to detect specific visual features such as edges or textures.
- **Stride:** The number of pixels the filter shifts across the input matrix during the convolution process. A stride of 1 means the filter moves one pixel at a time, while a stride of 2 skips every other pixel.
- **Padding:** Additional pixels (often zeros) added around the input image.
  - *Valid padding*: No extra pixels, reduced output size.
  - *Same padding*: Maintains original input dimensions.
- **Activation Function:** After convolution, an activation function such as ReLU is applied to introduce non-linearity into the model.

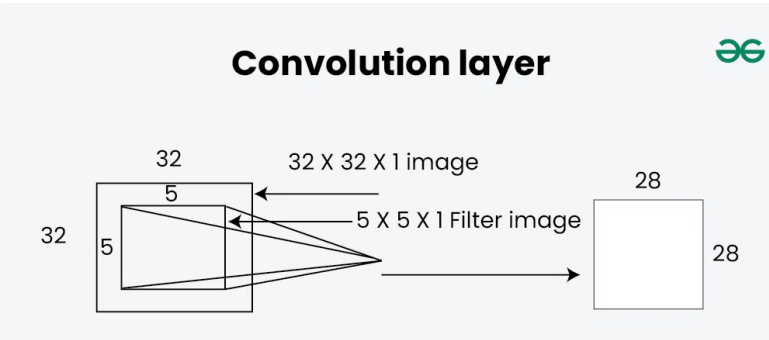


Figure 2.5: Convolution Layer Example with 32x32x1 Image and 5x5x1 Filter

Pooling Layers A pooling layer follows a convolutional layer and nonlinearity (e.g., ReLU) in a CNN. It downsamples each feature map using a 2×2 filter with stride 2, reducing the size by half (e.g., $6 \times 6 \rightarrow 3 \times 3$). Common methods are max pooling (takes maximum values) and average pooling (calculates averages), which improve efficiency and invariance.

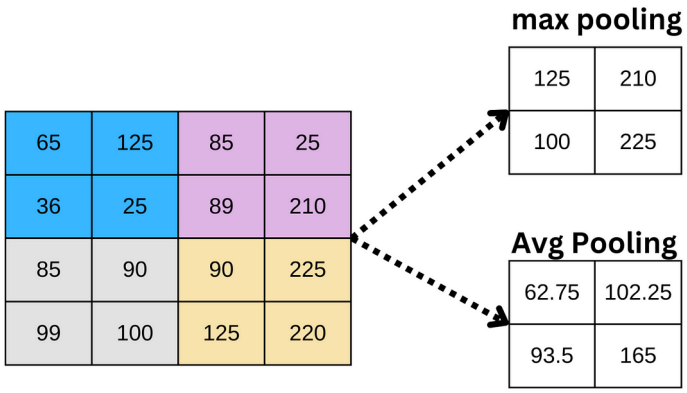


Figure 2.6: Two Pooling Methods Compared

Dropout During training, dropout randomly sets a fraction of the layer's neurons to zero with a probability "p" (known as the dropout rate). For instance, if the dropout rate is 0.5, half of the neurons will be deactivated in each training iteration. This ensures that the remaining neurons take on the responsibility of learning independently. (show figure) However, during inference (when the model is making predictions), dropout is turned off. Instead, the activations are scaled down by the same dropout rate to ensure consistency between training and testing phases [49].

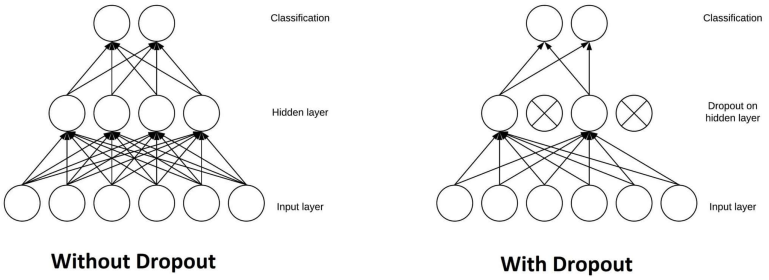


Figure 2.7: Comparing Neural Networks: With and Without Dropout

Fully Connected (Dense) Layers After a series of convolutional and pooling operations, the extracted features are flattened and passed through one or more fully connected layers. These layers combine the learned features and perform the final classification by assigning probabilities to each class.

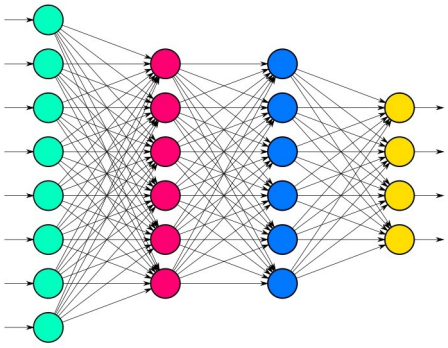


Figure 2.8: How a Dense Layer Connects Neurons

Visualizing CNN Workflow in Plant Disease Classification To better understand the role of CNNs in plant disease classification, the figure below illustrates the typical processing stages of a diseased leaf image through a CNN pipeline:

- **Convolutional + Activation Layers:** Extract edges, textures, and color variations.
- **Pooling Layers:** Reduce spatial dimensions and support generalization.
- **Flattening Layer:** Converts feature maps to a 1D vector.
- **Fully Connected Layers:** Perform the final classification.

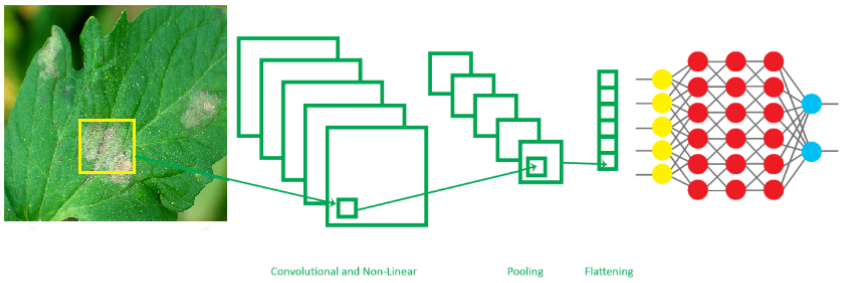


Figure 2.9: Schematic representation of CNN architecture applied to leaf disease detection

2.3.3 Transfer Learning in Plant Classification

Training deep CNNs from scratch requires large, annotated datasets—resources often unavailable in agriculture. Transfer learning addresses this limitation by reusing models pre-trained on large-scale datasets such as ImageNet, which contains over 14 million labeled images [50].

In this approach, the convolutional base of a pre-trained model like `MobileNetV2` or `EfficientNetB0` is used as a fixed feature extractor. A custom classification head—typically comprising a Global Average Pooling layer, a Dropout layer, and a Dense layer with softmax activation—is added for task adaptation.

Optionally, upper layers of the base model can be unfrozen for fine-tuning, allowing the model to adjust high-level features to the specific characteristics of diseased leaves.

This method not only reduces training time and computation costs but also improves performance, especially when data is limited. As a result, transfer learning has become standard practice in plant disease classification due to its efficiency and accuracy.

2.4 Related Work

2.4.1 Image Segmentation

The table above provides a comprehensive overview of recent research efforts in using advanced segmentation models to isolate plant leaves from complex backgrounds in agricultural image datasets. These studies highlight the diverse approaches and models employed to address the challenge of accurate leaf segmentation, which is crucial for improving the performance of plant disease detection models.

- **Model and Technique Variability**
  - The studies utilize a range of segmentation models, including U-Net with different backbones (EfficientNet, Inception, ResNet), DeepLabV3+, Feature Pyramid Networks (FPN), Mask R-CNN, and Mask Scoring R-CNN. The choice of model often depends on the specific requirements of the task, such as the complexity of the background, the type of crops, and the desired accuracy.
  - For instance, U-Net with EfficientNet backbone (Refs. 1, 7) is frequently used due to its efficiency and ability to handle binary segmentation tasks effectively. Similarly, U-Net with Inception and ResNet backbones (Refs. 5, 8) are employed for their robustness and accuracy in segmenting leaves from complex backgrounds.
  - Mask R-CNN and Mask Scoring R-CNN (Refs. 3, 6, 10) are used for more complex tasks involving disease segmentation in peach and tomato crops, demonstrating their capability to handle multi-class segmentation and object detection.
- **Crop Focus and Dataset Diversity**
  - The crop focus varies across the studies, with some concentrating on specific crops like tomato, wheat, peach, rice, corn, soybean, and barley, while others address multiple

crops. This diversity reflects the broad applicability of segmentation techniques across different agricultural contexts.

  - The datasets used are predominantly custom datasets, tailored to the specific needs of each study. This customization allows researchers to address the unique challenges posed by different crops and environmental conditions.
  - For example, the study on tomato leaf segmentation using U-Net with EfficientNet backbone (Ref. 1) utilizes the PlantVillage dataset, which is a well-known resource for agricultural image analysis.

- **Performance Metrics**

  - Performance metrics, such as Intersection over Union (IoU) and F1-score, are critical for evaluating the effectiveness of segmentation models. The studies report a range of performance metrics, with IoU and F1-score being the most commonly used.
  - For instance, the study on tomato leaf segmentation using U-Net with EfficientNet backbone (Ref. 1) reports an IoU of 0.97, indicating a high level of accuracy in segmenting tomato leaves from their backgrounds.
  - Similarly, the study on wheat leaf segmentation using DeepLabV3+ (Ref. 2) achieves an F1-score of 0.94, demonstrating the effectiveness of this approach in binary segmentation tasks.
  - The study on peach disease segmentation using Mask Scoring R-CNN (Ref. 3) reports an IoU of 0.86, highlighting the model's ability to accurately segment diseased areas.

- **Comparative Analysis**

  - Comparing the performance metrics across different models and crops, it is evident that U-Net with EfficientNet backbone achieves the highest IoU of 0.97 for tomato leaf segmentation (Ref. 1), while DeepLabV3+ achieves an F1-score of 0.94 for wheat leaf segmentation (Ref. 2).
  - These results suggest that U-Net with EfficientNet backbone is particularly effective for tomato leaf segmentation, possibly due to the high efficiency and accuracy of the EfficientNet backbone in handling complex backgrounds.
  - On the other hand, DeepLabV3+ performs well in wheat leaf segmentation, indicating its robustness in handling the specific challenges posed by wheat images.

- **Future Directions**

- The reviewed studies highlight several areas for future research. First, there is a need for more standardized datasets and performance metrics to facilitate direct comparison across different models and techniques.
- Second, the integration of advanced deep learning architectures, such as EfficientNet, ResNet, and Inception, could further enhance segmentation accuracy and efficiency.
- Third, the development of hybrid models that combine the strengths of multiple techniques may offer improved performance in complex agricultural environments.
- Finally, the application of these techniques to a broader range of crops and agricultural scenarios could significantly contribute to the advancement of precision agriculture.

In conclusion, the use of advanced segmentation techniques to isolate plant leaves from backgrounds has shown promising results in improving plant disease detection. The studies summarized in the table demonstrate the potential of various models and techniques, emphasizing the importance of dataset diversity and model selection. Future research should focus on standardizing evaluation metrics, exploring advanced architectures, and expanding the application of these techniques to diverse agricultural contexts.

Ref.	Model / Technique	Crop	Dataset	Classes	Task	Performance
[51]	U-Net (EfficientNet)	Tomato	PlantVillage	2	Leaf segmentation	IoU: 0.97
[52]	DeepLabV3+	Wheat	Custom	2	Leaf segmentation	F1: 0.94
[53]	Mask Scoring R-CNN	Peach	Custom	2	Disease segmentation	IoU: 0.86
[54]	FPN	Multi-crops	Custom	2	Leaf segmentation	IoU: 0.88
[55]	U-Net (Inception)	Rice	Custom	2	Leaf segmentation	Acc: 97%
[56]	Mask R-CNN	Peach	Custom	2	Disease segmentation	IoU: 0.86
[57]	U-Net (EfficientNet)	Corn	Custom	2	Leaf segmentation	Acc: 93%
[58]	U-Net (ResNet)	Soybean	Custom	2	Leaf segmentation	IoU: 0.88
[59]	DeepLabV3+	Barley	Custom	2	Leaf segmentation	IoU: 0.89
[60]	Mask Scoring R-CNN	Tomato	Custom	2	Disease segmentation	IoU: 0.87

Table 2.2. Overview of Image Segmentation Techniques in Plant Disease Detection

2.4.2 Image Classification

A thorough comparison of the different deep learning architectures used for plant disease detection across datasets is given in the table. The following are some important findings and observations:

1. **Variability in Architecture**

   - From more sophisticated models like EfficientNet, Xception, and Vision Transformers to more conventional CNNs like VGG16 and ResNet50, the table covers a broad spectrum of architectures. This variety is a reflection of the continuous development of deep learning methods as well as the requirement to modify models for particular tasks and datasets.
   - The application of hybrid models, like NASNetMobile and MobileNetV2 for feature fusion Entry 7, demonstrates the possible advantages of combining various architectures to capitalize on their advantages.

2. **Dataset Variability**

   - The size, complexity, and number of classes of the datasets used vary greatly. The PlantVillage dataset, for instance, is utilized in several entries with varying class counts Entries 3, 4, and 10, demonstrating its adaptability and appropriateness for evaluating various architectures.
   - Both the potato leaf disease dataset Entries 2 and 6 and the cassava leaf disease dataset Entries 1 and 5 are extensively used, indicating their significance in particular agricultural contexts.

3. **Performance Metrics**

   - The most often reported metric is accuracy, which can range from 87.7% to 99.85%. This broad range highlights how the architecture and the dataset affect model performance.
   - Some entries such as Entries 1, 3, and 8 report additional metrics, such as precision, recall, and F1-score, which offer a more nuanced picture of model performance. For example, Entry 1 exhibits balanced recall and precision, suggesting strong performance overall with little bias toward any class.
   - In contrast to the test accuracy (92.5%), Entry 12's high training and validation accuracy (99.5%) and 95.2%, respectively, raise the possibility of an overfitting problem, a frequent problem with deep learning models.

4. **Model Complexity and Efficiency**

- The need for accurate and efficient models, particularly in resource-constrained environments, is highlighted by the use of lightweight architectures such as MobileNetV2 Entries 7 and 9 and EfficientNet Entries 1, 4, and 5.
- Although it comes at the expense of greater complexity, the custom CNN with 28 layers and Fire Modules Entry 10 shows how customized architectures can achieve high accuracy.

In conclusion, the table illustrates the variety and intricacy of deep learning-based plant disease detection. It emphasizes how crucial it is to choose the right architectures and datasets while consistently coming up with new ideas to boost model effectiveness and performance.

2.5 Conclusion

This chapter has outlined the importance of segmentation and classification in plant disease detection systems. This chapter has thoroughly reviewed a variety of methods, and elaborated on the strengths of deep learning models in dealing with complex visual patterns observed in agricultural images. The related work section also addressed how these techniques have been leveraged in contemporary research, and thus established the precedent for the model design in the next chapter.

Ref	Architecture	Dataset	Classes	Performance Metrics	Year
[61]	EfficientNet-B3, InceptionV3, ResNet50, VGG16	Cassava leaf disease dataset	5	Accuracy: 87.7%; Precision: 87.8%; Recall: 87.8%; F1-score: 87.7%	2025
[62]	DenseNet201	Potato leaf disease dataset	10	Training Accuracy: 99.5%; Validation Accuracy: 95.2%; Test Accuracy: 92.5%	2024
[63]	Xception, ULEN, MobileNet_v2, ShuffleNet_v2, VGG16	PlantVillage	15	Xception Precision: 99.23%; Recall: 99%; F1-score: 0.991	2023
[64]	EfficientNet	PlantVillage (38 classes)	38	Accuracy: 95%	2025
[65]	EfficientNet-B3, InceptionV3, ResNet50, VGG16	Cassava leaf disease dataset	4	Accuracy: 87.7%; Precision: 87.8%; Recall: 87.8%; F1-score: 87.7%	2025
[66]	DenseNet201	Potato leaf disease dataset	3	Accuracy: 92.5%	2024
[67]	MobileNetV2 + NASNetMobile (Feature Fusion)	Corn & Coffee Leaf Datasets	6	97.53% (Corn), 94.65% (Coffee) Accuracy	2023
[68]	VGG16, InceptionV3, DenseNet20 + Vision Transformer	Apple & Corn Datasets (4 classes each)	8	99.24% (Apple), 98% (Corn) Accuracy	2025
[69]	MobileNetV2	Plant Pathology Dataset	5	98.71% Accuracy	2024
[70]	Custom CNN (28 layers with Fire Modules)	PlantVillage (8 classes, ∼2,080 images)	8	98.49% (8-class), 99.85% (3-class) Accuracy	2023

Table 2.3: Comparison of Plant Disease Detection Architectures

Chapter 3

Proposed System Architecture and Methodology

3.1 Introduction

This chapter discussed the datasets, preprocessing methods, and model architectures that were selected for segmentation and classification problem formulation. The goal is to build competent model systems specifically for plant disease detection, explicitly in the context of established frameworks of deep learning. This has included model selection, information preparation, and training plans.

3.2 Dataset Description and Preprocessing

3.2.1 Dataset Overview

We utilized two distinct datasets in our study: one for image classification and another for image segmentation. Below are detailed statistics for each dataset presented in tabular format.

3.2.1.1 Classification Dataset Overview

The classification dataset includes 14,463 labeled plant leaf images across 15 classes, formatted for training deep learning models. All images are in JPG format with consistent dimensions.

Class	Plant Name	Disease Name	Number of Images
C1	Strawberry	Healthy	319
C2	Strawberry	Leaf Scorch	776
C3	Tomato	Bacterial Spot	1,488
C4	Tomato	Early Blight	699
C5	Tomato	Healthy	1,113
C6	Tomato	Late Blight	1,335
C7	Tomato	Leaf Curl	3,749
C8	Tomato	Leaf Mold	666
C9	Tomato	Mosaic Virus	261
C10	Tomato	Septoria Leaf Spot	1,239
C11	Tomato	Spider Mites	1,173
C12	Tomato	Target Spot	982
C13	Watermelon	Anthracnose	108
C14	Watermelon	Downy Mildew	265
C15	Watermelon	Mosaic Virus	290
Total Images			**14,463**

Table 3.1: Classification Dataset Summary

Sample Images : The figure presents representative examples from the classification dataset, illustrating the visual diversity across different plant species and disease classes. Each image depicts a

leaf labeled with its corresponding class, including both healthy and diseased conditions. The samples showcase various symptoms such as discoloration, spots, curling, and fungal growth, which are critical features used by the classification models to distinguish between classes. This visual diversity emphasizes the importance of a robust model capable of generalizing across different plant types, disease patterns, and imaging conditions.



(a) Tomato late blight



(b) Tomato Septoria spot



(c) Strawberry leaf scorch



(d) Tomato healthy



(e) Tomato early blight



(f) Tomato target spot

Figure 3.1: Visual examples of plant diseases analyzed in this study.

3.2.1.2 Segmentation Dataset Overview

This section provides an overview of the segmentation dataset used to train and evaluate models for isolating plant leaves from the background. The dataset consists of RGB images of individual leaves along with binary masks that highlight only the leaf regions, excluding soil, shadows, and other background elements.

Dataset Description : The segmentation dataset includes 233 RGB images of plant leaves and 233 corresponding binary masks, for a total of 466 files. Each RGB image contains a single leaf against a complex background, while the mask provides pixel-wise annotation of the leaf area.

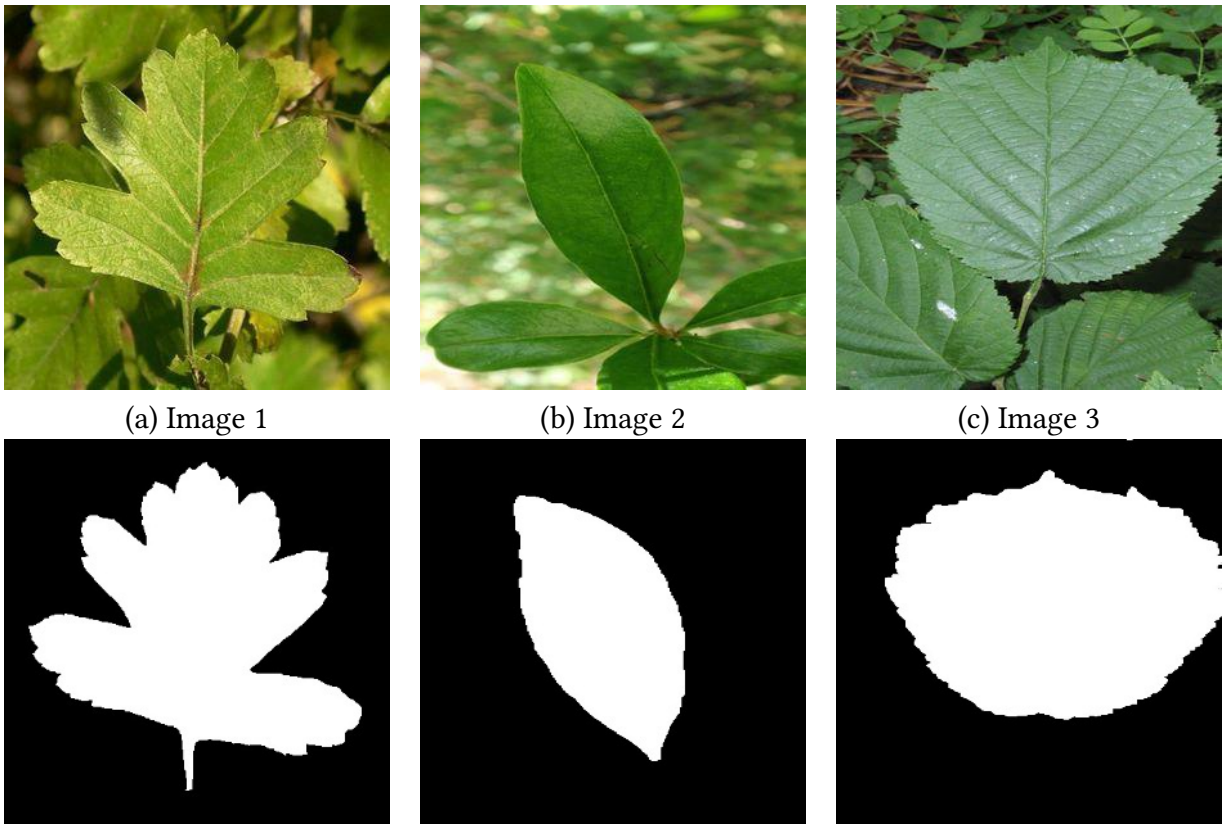
The goal of this segmentation task is to separate the leaf region from the surrounding background. This preprocessing step is crucial for subsequent tasks, such as disease classification, by ensuring the model focuses on the relevant plant area without being affected by background noise.

Dataset Characteristics : The table summarizes the key attributes of the segmentation dataset, including file format, image dimensions, and file size statistics.

Attribute	Value
Total Images	466 (233 images + 233 masks)
Image Format	.jpg
Image Dimensions (min)	179 × 297
Image Dimensions (max)	600 × 800
Image Dimensions (mean)	321 × 345
File Size Range	5.32 KB – 34.17 KB
File Size Mean	13.96 KB

Table 3.2: Segmentation Dataset Summary

Sample Images : The Figure shows representative examples from the segmentation dataset. Each row displays an original RGB image of a plant leaf alongside its corresponding binary mask. The masks highlight the precise shape and boundaries of the leaf, enabling models to learn how to extract leaf regions from various backgrounds.



(a) Image 1 (b) Image 2 (c) Image 3

(d) mask 1 (e) mask 2 (f) mask 3

Figure 3.2: Visual examples of Segmentation Dataset

3.2.2 Preprocessing Techniques

Before training the models, we conducted several preprocessing steps on the dataset to ensure everything was clean and homogeneous. All images were resized to a constant dimension appropriate for each model and normalized to the range$[0, 1]$ in order to allow efficient learning.

We also included data augmentation techniques during training as a method of promoting generalization and reducing the risk of overfitting. Data augmentation in this experiment included horizontal flipping, rotation, and zoom.

```
train_gen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    zoom_range=0.2,
    horizontal_flip=True,

)
```

Figure 3.3: Preprocessing and Augmentation Parameters

In classification tasks, we one-hot encoded the categorical labels for compatibility with the softmax output of the models. Additionally, to address the issue of class imbalance within the dataset—where certain disease classes were underrepresented—we applied targeted data augmentation to increase the presence of minority classes and improve their contribution during training. We also computed and applied class weights during the training process to ensure that the model gave proportionally more importance to underrepresented classes. These combined strategies helped ensure more balanced learning and reduced bias toward overrepresented categories.

```
labels = train_data.classes
class_weights = compute_class_weight(
    class_weight='balanced',
    classes=np.unique(labels),
    y=labels
)
class_weights = dict(enumerate(class_weights))
```

Figure 3.4: Computing Class Weights to Handle Imbalanced Dataset

3.2.3 Dataset Splitting

To ensure a fair and appropriate assessment of model performance, we divided each dataset into three subsets: training set, validation set, and test set. For the classification task, we allocated 70% of the data to the training set, while the remaining 30% was split into 10% for validation and 20% for testing. This strategy allowed the model to learn from a large portion of the data while retaining sufficient examples for unbiased evaluation on unseen data.

For the segmentation task, we followed a similar protocol but ensured that each subset contained a representative portion of all disease classes. This step was crucial to maintain the validity of the evaluation and ensure generalization across different types of infections.

The validation set played an essential role during training, enabling real-time evaluation of model performance. It facilitated the use of techniques such as early stopping and model checkpointing, which helped to prevent overfitting by reverting to the best-performing model during the training process. This not only improved training efficiency but also provided a more accurate picture of the model's learning progress.

```
random.shuffle(files)

   n_total = len(files)
   train_split = int(0.7 * n_total)
   test_split = int(0.9 * n_total)

   train_files = files[:train_split]
   test_files = files[train_split:test_split]
   val_files = files[test_split:]
```

Figure 3.5: Python Script for Splitting Dataset into Train, Test, and Validation Sets

3.3 Model Selection Criteria

Several factors influenced the choice of models used in this work:

- **Lightweight Architectures for Mobile Deployment:** Since the final system is intended to run on mobile devices via a Flutter application, models with low memory consumption and fast inference time were prioritized. Lightweight architectures are essential to meet the constraints of mobile deployment environments.
- **Pretrained Models (Transfer Learning):** To reduce training time and improve performance with limited data, we utilized pretrained models through transfer learning. These models

benefit from rich feature extraction capabilities learned from large-scale datasets .

- **Performance vs. Complexity:** While high-performance models are generally desirable, excessively complex architectures can be impractical for real-time or resource-constrained applications. Therefore, we focused on selecting models that strike a balance between classification accuracy and computational cost.

3.4 Segmentation Models Description

3.4.1 Overview of Segmentation Approach

In this research, we implemented a semantic segmentation methodology to differentiate plant leaves from the background. Semantic segmentation is the allocation of a class label (either foreground or background) to every pixel of an image so that a plant structure can be defined down to its pixels [71]. This methodology is important for the analysis of plant diseases since it allows the classification stage to process only the relevant leaf regions while eliminating or diminishing the background noise produced by soil, shadows, or objects.

By narrowing the classifier's focus to segmented plant regions, segmentation improves the accuracy, robustness, and generalization of the disease classification model. It also helps reduce false positives and allows for more consistent feature extraction under varying environmental conditions [33].

3.4.2 Model Architectures

Several contemporary deep learning models were evaluated for semantic segmentation. All architectures follow an encoder–decoder structure and were initialized with pre-trained weights to leverage transfer learning.

- **U-Net with EfficientNetB0 Backbone**
  Structure: U-Net architecture using EfficientNetB0 as encoder with a symmetric decoder and skip connections.
  Characteristics: Optimized for scale efficiency with compound scaling, lightweight and edge-deployable.
  Initialization: Pre-trained on ImageNet [72].

- **U-Net with InceptionV3 Backbone**
  Structure: U-Net encoder based on InceptionV3.
  Characteristics: Inception modules allow multi-scale feature extraction within a single block; capable of learning both detailed and abstract patterns.
  Initialization: Pre-trained on ImageNet [73].

- **U-Net with ResNet34 Backbone**
  Structure: Standard U-Net decoder attached to a ResNet34 encoder.
  Characteristics: Residual connections enhance gradient flow and training efficiency; balances performance and complexity.
  Initialization: Pre-trained on ImageNet [15].

- **Feature Pyramid Network (FPN) with ResNet50 Backbone**
  Structure: Combines bottom-up and top-down paths with lateral connections across scales.
  Characteristics: Enables effective segmentation of objects at multiple sizes using hierarchical multi-scale features.
  Initialization: Pre-trained on ImageNet [74].

Each model presents unique strengths: U-Net variants are efficient and interpretable; FPN excels at multi-scale object segmentation.

3.4.3 Preprocessing and Input for Segmentation

Uniform preprocessing steps were applied across all datasets to ensure high-quality model training:

- **Input Size:** All images and masks resized to 256×256 pixels for standardization and computational efficiency.
- **Normalization:** Input images scaled to the $[0, 1]$ range via min-max normalization (pixel / 255).
- **Mask Preparation:** Binary segmentation format with grayscale masks where 0 = background, 1 = plant.
- **Augmentation:** Random horizontal/vertical flips, rotation ($\pm 15^{\circ}$), zoom, and brightness adjustments to reduce overfitting and improve generalization [75].
- **Data Pairing:** Maintained one-to-one correspondence between original images and masks for supervised training.

3.4.4 Ensemble Strategy

To improve robustness and mitigate individual model bias, ensemble learning techniques were explored [76]. These techniques integrate multiple predictions to produce more stable segmentation outputs. The strategies investigated include:

- **Weighted Average:** Merges probability maps from multiple models using predetermined weights. Thresholding converts the result into a binary mask.

- **Majority Voting:** Converts all outputs to binary masks and classifies a pixel as foreground if the majority of models agree.
- **Stacked Ensemble:** A shallow CNN is trained to combine the predictions from base models for final segmentation.

These ensemble methods leverage the diverse capabilities of individual architectures, enhancing generalization and consistency, particularly in complex or ambiguous regions.

3.5 Classification Models Description

In this section, we describe the classification models used in our experiments, highlighting their architectures, justification for their selection, and the specific training configurations applied.

3.5.1 MobileNetV2

MobileNetV2 is a convolutional neural network architecture specifically designed for mobile and embedded devices with limited computing resources and memory. It was introduced to address the challenges of deploying traditional convolutional neural networks (CNNs) in constrained environments.

The core innovation of MobileNetV2 is the use of **inverted residual blocks with linear bottlenecks**. Unlike conventional architectures that rely on wide representations, MobileNetV2 connects layers using both low- and high-dimensional representations. Each block expands the input using a 1×1 convolution, applies an efficient 3×3 depthwise convolution, and then compresses the result with a final 1×1 linear convolution. Importantly, non-linear activations are omitted in the narrow (bottleneck) layers to preserve high-dimensional information.

The architecture begins with a standard convolutional layer with 32 filters, followed by 19 inverted residual bottleneck blocks. In its standard configuration (with a width multiplier of 1.0 and input size of 224×224), the model has approximately 3.4 million parameters and requires around 300 million multiply-add operations (MAdds).

This design enables MobileNetV2 to balance efficiency and accuracy, making it particularly suitable for real-time applications on mobile and embedded platforms [77].

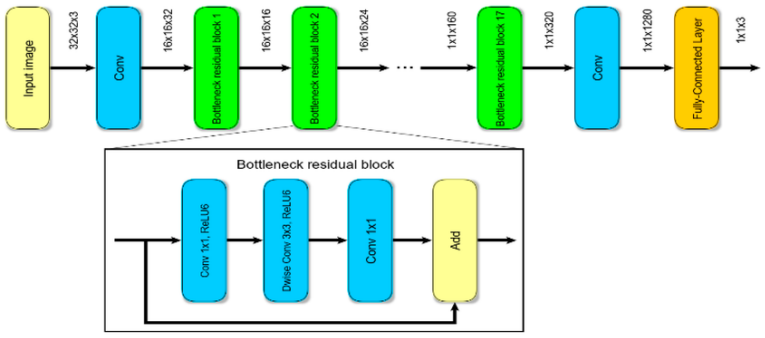


Figure 3.6: architecture of MobileNetV2

3.5.2 EfficientNetB3

EfficientNet-B3 is a convolutional neural network (CNN) model of the EfficientNet family, as described by Tan and Le in 2019. EfficientNet models achieve high accuracy and performance with low resource consumption via a compound scaling approach that scales the depth, width, and input resolution of networks. Compared to EfficientNet-B0, EfficientNet-B3 has scaling coefficients of width=1.2, depth=1.4, and increased input resolution of 300×300 pixels. The EfficientNet-B3 model is composed of 9 stages, starting with a 3×3 convolutions block, followed by MBConvs (Mobile Inverted Bottleneck Convolutions) blocks with 3×3 or 5×5 kernel, expansion ratio and inverted bottleneck, culminating in a 1×1 convolution; pooling and fully-connected layers are included in the model as well. The EfficientNet-B3 model has approximately 12 million trainable parameters, and achieves top-1 accuracy of 81.6% on ImageNet, due in part, to an excellent performance-resource trade-off[78]



Figure 3.7: EfficientNetB3 architecture

3.5.3 VGG16

VGG16 is a well-known convolutional neural network (CNN) model that has contributed greatly to varying computer vision tasks. CNNs are a category of artificial neural networks meant to process visual data and interpret visual data the same way as humans interpret images. VGG16 was created by the Visual Geometry Group at the University of Oxford and has a deep but simple design. VGG16 employs a predefined architecture for input, output, and hidden layers, where the depth of the architecture is 16 weight layers. One of the main design characteristics of VGG16 is to use small

(3 × 3) convolution filters so that the network can be increased in depth and yet maintain use of computation resources. The VGG16 model has roughly 138 million trainable parameters, thus allowing the model to learn complex attributes and features of images. When VGG16 completed the ImageNet dataset, the model achieved a top-5 accuracy of 92.7% across 1,000 different image categories. VGG16 is still one of the most widely used models in image classification or image detection due to its accuracy and suitability for transfer learning [79] .

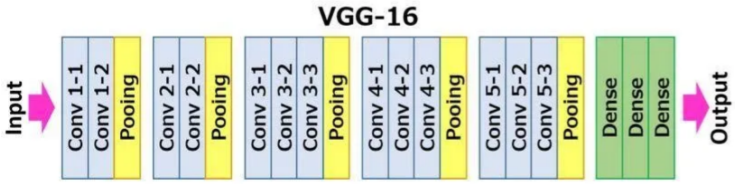


Figure 3.8: VGG16 architecture

3.6 Project Requirements

This section outlines the essential tools, programming languages, and libraries used in building and training the segmentation and classification models.

3.6.1 Development Environment

The training and testing of the deep learning models were conducted using Google Colab, a cloud-based platform that provides free access to computing resources. Given the nature of our models and the size of the dataset, we primarily utilized the CPU runtime in Colab. The typical hardware configuration available in this environment includes:

- **Processor:** Single-core hyper-threaded Xeon Processor @ 2.3 GHz (1 core, 2 threads)
- **RAM:** Approximately 12.6 GB available
- **Disk Storage:** Approximately 33 GB available

3.6.2 Programming Languages

Python: Python is a high-level, interpreted, and object-oriented programming language with dynamic semantics. Since the language has built-in data structures, dynamic typing, and dynamic binding [80], Python is especially good for rapid development. For example, if you are currently working on a machine learning project, it is not uncommon to need to switch up the methodology

you are implementing three or four times throughout the process. Python is commonly utilized in domains such as data science, machine learning, and deep learning because of its ease of use. The emphasis on readability of Python's syntax leads to clarity when reading and writing the code itself, which can help to significantly reduce development and maintenance costs. Whereas other programming languages may require extensive coding to create and deploy a machine learning or deep learning model, Python's inclusion of vast libraries and frameworks (examples include TensorFlow, PyTorch, feeding a recurrent neural network through scikit-learn, etc.) leads to a more seamless coding experience. Both the Python interpreter and the extensive standard library are free to download in both source code and binary form for most platforms.

3.6.3 Deep Learning and Machine Learning Libraries

Several essential libraries and tools were employed for implementing and managing different parts of the project:

- **TensorFlow:** TensorFlow is an open-source machine learning (ML) and AI framework built by Google Brain. It was created to make building machine learning models — specifically deep learning models — easier and to help you build, train, and deploy ML models across a variety of platforms. TensorFlow provides all the building blocks to develop a variety of ML applications including natural language processing (NLP), computer vision (CV), time series forecasting, and reinforcement learning [81].

- **Keras:** Conversely, Keras is a Python-based high-level neural network API, and it is an interface for TensorFlow and other backends like Theano and CNTK. Keras was designed to be user-friendly, modular, and extensible to enable fast and easy prototyping of deep learning models.

- **TensorFlow Lite (TFLite):** TensorFlow Lite is a specific framework in the TensorFlow ecosystem that allows you to run machine learning models directly on mobile, embedded, and edge devices. TFLite has been designed to be lightweight and efficient for deployment on platforms with limited compute power and memory capacity. With TFLite, operations are designed for on-device inference, offering an ideal solution for real-time applications such as image recognition and object detection.

- **NumPy:** (short for Numerical Python) is a core library for performing numerical computations on large datasets. NumPy provides support for multi-dimensional arrays and matrices as well as a large library of high-level mathematical functions to operate on these arrays. NumPy arrays are faster and more flexible than Python lists, enabling faster element access and efficient data manipulation over wired and live data [82].

- **Pandas:** Pandas is a Python library used for data manipulation and analysis. It provides a convenient way to analyze and clean data. The Pandas library introduces two new data structures to Python — Series and DataFrame — both built on top of NumPy.
- **Matplotlib:** Matplotlib is a Python library used for plotting graphs with the help of other libraries like NumPy and Pandas. It is a powerful tool for visualizing data in Python and is commonly used for creating statistical inferences and plotting 2D graphs.
- **Seaborn:** Seaborn is a powerful library for creating data visualizations in Python, built on top of Matplotlib. It provides an easy, high-level way of creating attractive and informative statistical graphics. Seaborn integrates closely with Pandas and its data structures, facilitating the use of DataFrames and Series. It allows simpler visualization of complex datasets, helping to understand data distribution and relationships between variables.
- **scikit-learn:** Scikit-learn is a comprehensive Python library for data analysis and model evaluation. It leverages basic packages such as NumPy, SciPy, and Matplotlib to provide a suite of reliable and efficient tools for computing evaluation metrics including accuracy, precision, recall, F1-score, and Matthews Correlation Coefficient (MCC). Though deep learning is used in this project, scikit-learn remains valuable for assessing and validating model performance with standard statistical measures.
- **OpenCV:** OpenCV (Open Source Computer Vision Library) is an established open-source library with many capabilities in computer vision and image processing. It is flexible and efficient, supporting image filtering, feature extraction, object detection, and morphological transformations. OpenCV's extensive capabilities make it an easy-to-use support library for preprocessing and analyzing visual data, including deep learning applications [83].
- **Segmentation Models :** The `segmentation_models` library is a popular open-source toolkit for building image segmentation models. It offers a wide range of state-of-the-art architectures, such as U-Net, LinkNet, FPN, and PSPNet, with pretrained backbones like ResNet, EfficientNet, and VGG. This library simplifies the process of developing and training segmentation networks by providing a clean, user-friendly API compatible with PyTorch. It supports multi-class segmentation, loss functions, metrics, and GPU acceleration, enabling efficient experimentation and deployment of semantic segmentation models [84].

3.7 Evaluation Metrics

To rigorously assess the performance of the proposed system, appropriate evaluation metrics were employed for both the segmentation and classification tasks. Each set of metrics was selected based

on its relevance and effectiveness in capturing the performance characteristics of the respective task.

3.7.1 Segmentation Metrics

For the segmentation component, evaluation was conducted using two widely accepted metrics: the F1 Score (also referred to as the Dice Coefficient) and the Intersection over Union (IoU). These metrics were computed during the testing phase to quantify the overlap between predicted masks and ground truth annotations.

- **F1 Score (Dice Coefficient):** The F1 Score, also known in the context of image segmentation as the Dice Coefficient, is a harmonic mean of precision and recall. It evaluates the similarity between the predicted segmentation and the ground truth by measuring their overlap. A higher F1 Score indicates a stronger agreement between the predicted and actual segmentations. Mathematically, it is defined as:

$$\text{F1 Score (Dice)} = \frac{2 \times |P \cap G|}{|P| + |G|}$$

  where P and G represent the set of pixels in the predicted and ground truth masks, respectively.

- **Intersection over Union (IoU):** IoU, also known as the Jaccard Index, measures the proportion of the intersection area over the union area of the predicted and ground truth segmentation masks. It provides a robust estimate of the model's ability to correctly localize and segment the object of interest. It is defined as:

$$\text{IoU} = \frac{|P \cap G|}{|P \cup G|}$$

3.7.2 Classification Metrics

To evaluate the performance of the classification model, several standard metrics were utilized, including Accuracy, Precision, Recall, F1 Score, and the Matthews Correlation Coefficient (MCC). These metrics provide a comprehensive evaluation of classification performance, especially in scenarios involving class imbalance.

- **Accuracy:** Accuracy measures the overall correctness of the classifier by computing the ratio of correctly predicted instances (true positives and true negatives) to the total number of instances. It is expressed as:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

- **Precision:** Precision quantifies the proportion of correctly predicted positive instances among all instances predicted as positive. It is calculated as:

$$\text{Precision} = \frac{TP}{TP + FP}$$

- **Recall:** Recall, also known as sensitivity or true positive rate, evaluates the ability of the model to identify all relevant positive cases. It is defined as:

$$\text{Recall} = \frac{TP}{TP + FN}$$

- **F1 Score:** The F1 Score is the harmonic mean of Precision and Recall, providing a single measure that balances both concerns. It is particularly useful in situations where the dataset is imbalanced. The formula is:

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

- **Matthews Correlation Coefficient (MCC):** The MCC is a correlation coefficient that takes into account all four confusion matrix categories and returns a value between −1 and +1. A value of +1 indicates a perfect prediction, 0 indicates no better than random prediction, and −1 indicates total disagreement. It is especially effective for imbalanced datasets. The MCC is computed as:

$$\text{MCC} = \frac{(TP \times TN) - (FP \times FN)}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

3.8 Conclusion

In this chapter, we introduced the principal elements of the experimental environment for our project. We began by presenting the dataset used for both segmentation and classification tasks, including its organization and preprocessing steps. Next, we provided an overview of the deep learning models employed, detailing their architectures and the training processes involved.

Following this, we described the main tools, libraries, and programming languages—primarily Python and its rich ecosystem—that facilitated efficient development and evaluation of our models. All these components are integrated into a comprehensive deep learning pipeline, which forms the foundation of our project workflow.

In the next chapter, we will present the performance of the deep learning models, analyzing their results with detailed evaluation metrics, illustrative figures, and comparative tables. Additionally, we will offer a brief overview of the mobile application development process and discuss the deployment of the most effective model within this application.

Chapter 4

Model Evaluation and Application Deployment

4.1 Introduction

In this chapter, we present a comprehensive evaluation of the deep learning models developed for the segmentation and classification of plant diseases. The evaluation is structured in two main sections: leaf segmentation and disease classification. We begin by assessing the performance of the segmentation model, which is designed to accurately isolate the leaf area from the background in plant images. This step is crucial for enhancing the quality of the input data used by the classification model. By segmenting the leaf region, we aim to reduce background noise and improve the precision of disease detection. The effectiveness of this preprocessing phase is then analyzed in relation to its impact on the subsequent classification performance. After covering the results from segmentation we will describe the results from classification. The classification models are compared and contrasted between the models that are trained without segmentation (using raw images) compared to the models that are trained on segmented images. This helps us critically appraise the extent to which segmenting images improves the detection performance for plant diseases.

The last part of this chapter will describe how we deployed the appropriate models for integration with a real-world mobile application. We describe the process of converting the models to TensorFlow Lite (TFLite) format, to embed into a Flutter based app, and how we built the real-time inference pipeline. The mobile development process was designed to be quick and simple to facilitate ease of use for farmers and agriculture professionals' access to fast in-field plant disease diagnosis.

4.2 Model Evaluation and Discussion

In this section, we first highlight why image segmentation is important as a preprocessing step for plant disease classification. The leaf images were first segmented so that the regions of the leaf with the plant would be separated from the background, i.e., the classification models were only trained on the regions of the image that were relevant to the plant leaf. Then we provided a thorough analysis of the classification results of three Deep Learning models. Each model was evaluated under two experimental conditions: (1) segmentation bounding boxes, where the regions of plant were separated, and (2) raw leaf images. The main purpose of this comparison is not aimed at delivering the best model to deploy but to evaluate the role of segmentation on performance and establish the best combination of model and input type (segmentation vs raw) that provides the most accurate classification for plant disease detection.

4.2.1 Segmentation Model Results and Analysis:

This section presents the performance evaluation and comparative analysis of four segmentation models used for plant leaf segmentation. All models were implemented using the `segmentation_models` Keras library:

- U-Net with EfficientNetB0 encoder
- U-Net with ResNet34 encoder
- U-Net with InceptionV3 encoder
- Feature Pyramid Network (FPN) with ResNet50 encoder

Each model was trained and evaluated on the same dataset using consistent performance metrics: **F1 Score**, **Intersection over Union (IoU)**, and **Loss**. Results were reported on the training, validation, and test sets to evaluate both learning capacity and generalization.

4.2.1.1 Quantitative Results

Figures 4.1, 4.2, and 4.3 display the comparison of F1 Score, IoU, and Loss values for all models across the training, validation, and test sets.

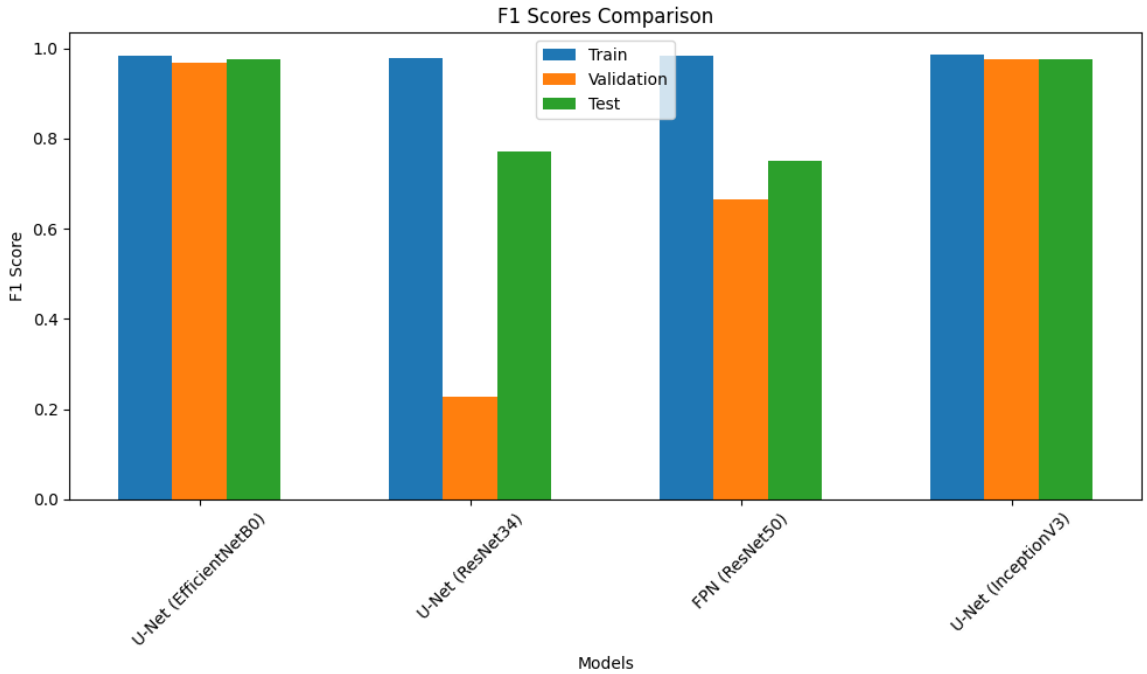


Figure 4.1: F1 Score comparison for all models on train, validation, and test sets.

Observations: The U-Net models with InceptionV3 and EfficientNetB0 encoders achieved the highest F1 scores on the test set (~0.975). U-Net with ResNet34 showed strong overfitting, with a validation F1 score of only 0.227.

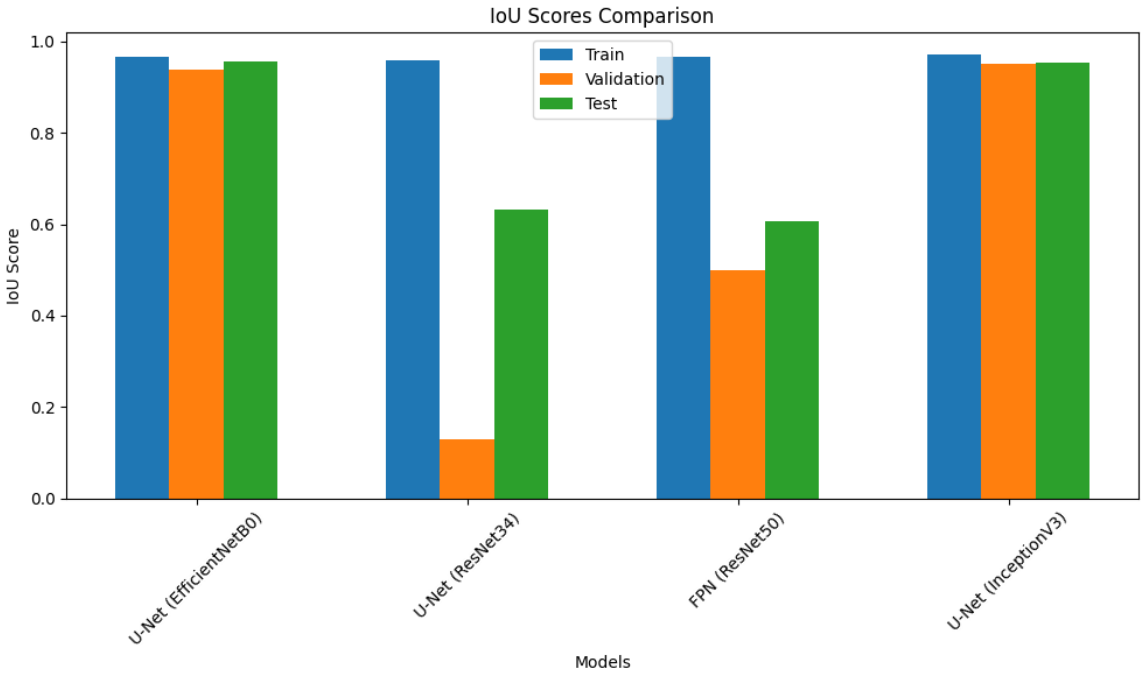


Figure 4.2: IoU Score comparison for all models on train, validation, and test sets.

Observations: InceptionV3 and EfficientNetB0 encoders maintained high IoU values across all datasets. ResNet34 underperformed on validation and test data, with a validation IoU of only 0.128.

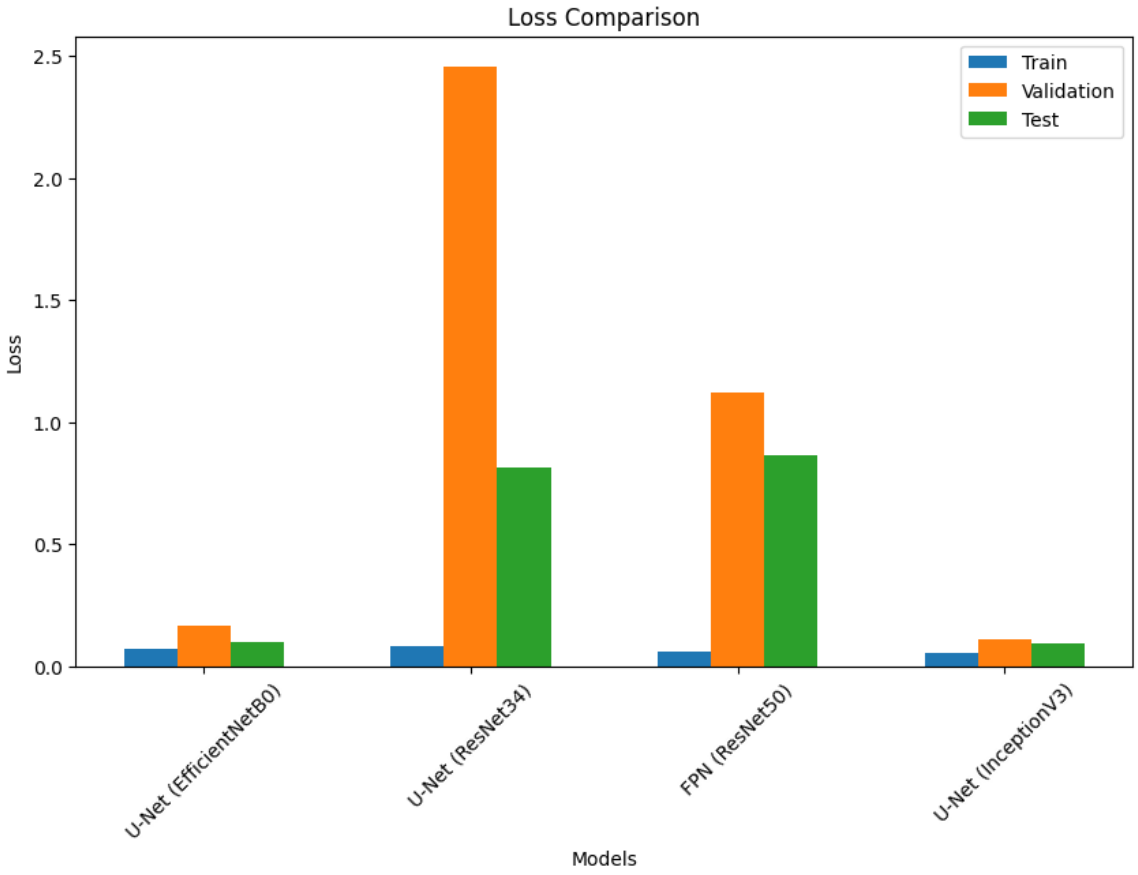


Figure 4.3: Loss comparison for all models on train, validation, and test sets.

Observations: U-Net with InceptionV3 exhibited the lowest loss values on validation (0.1102) and test sets (0.0934), indicating stable training. The ResNet34 model presented high validation loss (2.4580), confirming poor generalization.

4.2.1.2 Comparative Analysis

Based on the presented results:

- **Best Model: U-Net (InceptionV3)** – demonstrated the highest F1 and IoU scores and lowest loss across all sets.
- **Runner-up: U-Net (EfficientNetB0)** – strong performance and good generalization.
- **Moderate Performance: FPN (ResNet50)** – acceptable results but lower than the top two.
- **Underperforming: U-Net (ResNet34)** – exhibited overfitting and instability across validation and test data.

The assessment demonstrated that across the segmentation models analysed, U-Net with InceptionV3 and U-Net with EfficientNetB0 encoders outperformed the other models in all the evaluation

metrics examined (F1 Score, IoU, and Loss). Both models exhibited strong generalisation ability and stable training performance on validation and test sets. In contrast, the U-Net with ResNet34 encoder, showed signs of overfitting and poor generalisation ability, whereas the FPN with ResNet50, exhibited modest performance overall. These results provide useful insight and direction on the selection of segmentation models in plant disease detection pipelines, especially if extracting an accurate representation of the true leaf region is important for improving overall performance classification performance.

4.2.2 Classification Model Results and Analysis

Each model's performance is presented in two scenarios: without segmentation and with segmentation. For both cases, training and test plots are provided, followed by evaluation metrics and analysis.

4.2.2.1 Training Configuration

To ensure consistency and fairness across all classification models, a unified training configuration was employed. This allowed us to evaluate each model's performance under the same conditions.

Training Hyperparameters

Parameter	Value
Optimizer	Adam
Initial Learning Rate	0.0001
Loss Function	Categorical Crossentropy
Epochs	20
Batch Size	32

Table 4.1: Training Hyperparameters Used Across All Classification Models

Learning Rate Scheduling and Callbacks To improve model convergence and mitigate overfitting, the following callbacks were integrated during training:

- **Early Stopping**: Training is stopped if the validation loss does not improve for 5 consecutive epochs.
  `monitor='val_loss', patience=5`
- **Model Checkpoint**: The best model (based on validation loss) is saved.
  `save_best_only=True`

- **ReduceLROnPlateau**: Reduces the learning rate by a factor of 0.2 if no improvement in validation loss is observed for 3 epochs.
  `factor=0.2, patience=3`

Transfer Learning Strategy To take advantage of pretrained knowledge and to reduce training time on our dataset, transfer learning was employed:

1. All models were initialized with pretrained weights from ImageNet.
2. The convolutional base (feature extractor) was frozen during the initial training phase.
3. Once the validation performance stabilized, selected top layers of the base model were unfrozen.
4. Fine-tuning was performed using a lower learning rate to improve accuracy while preventing catastrophic forgetting.

4.2.2.2 MobileNetV2 Results

I. **Without Segmentation** : In the experiment the MobileNetV2 model was trained using the original unsegmented plant disease dataset and an input image size of 224×224, and a batch size of 32, for 20 epochs. The model showed consistent improvement in terms of performance through the training process. The resulting training accuracy was approximately 99.60% with a validation accuracy of 97.41% .The resulting training and validation loss was 0.098 and 0.077 respectively. The plots below show the training and validation accuracy and loss during the 20 epochs of training.

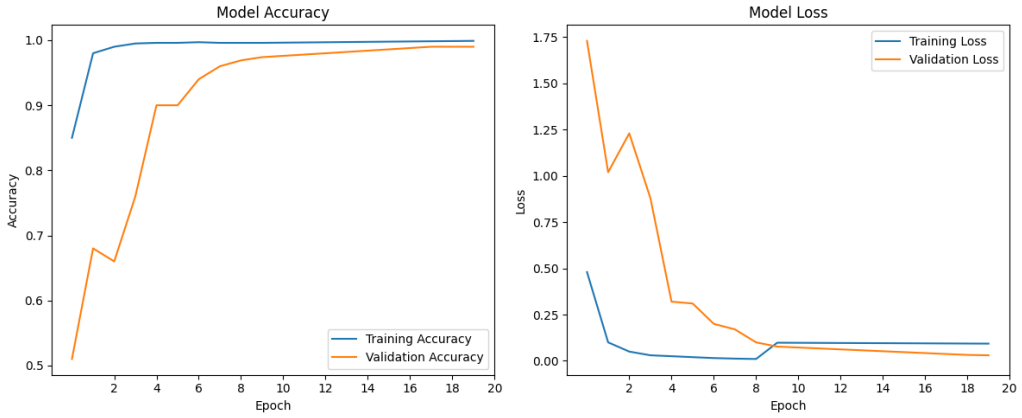


Figure 4.4: Training and Validation Accuracy and Loss of MobileNetV2 without Segmentation

After training, the model was evaluated using the test dataset with a resulting test accuracy of 97.43%. Also, a complete evaluation was carried out using the performance metrics of f1 score, precision, recall, and MCC.

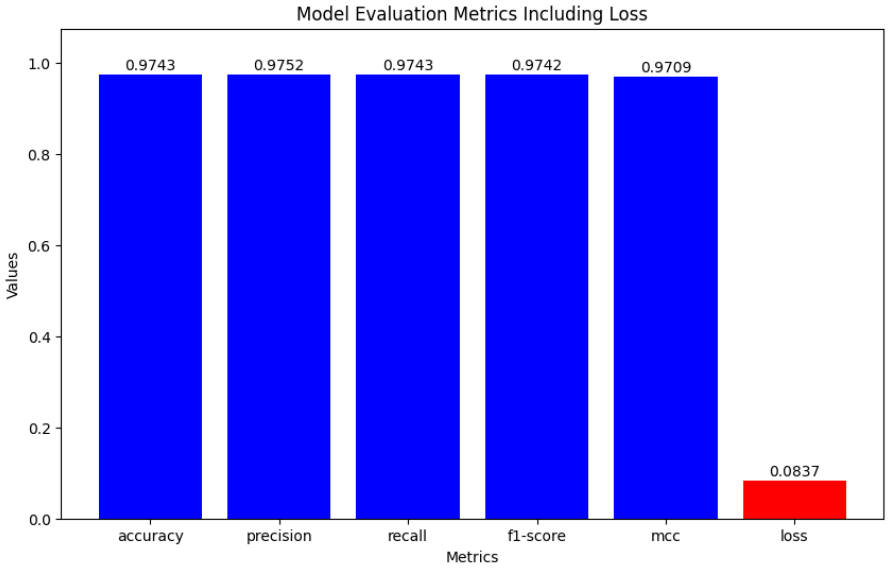


Figure 4.5: Model Metrics Overview on Test Set Without Segmentation

II. **With Segmentation** : In the experiment, the MobileNetV2 model was trained using the segmented plant disease dataset with an input image size of 224×224 and a batch size of 32. The training was conducted for a maximum of 20 epochs with early stopping enabled to prevent overfitting. The model demonstrated a steady improvement in performance during training. Early stopping was triggered when the validation loss stopped decreasing, ensuring optimal generalization. The final training accuracy reached approximately 99.73%, while the validation accuracy was 98.02%. The corresponding training and validation loss values were 0.061 and 0.053, respectively. The plots below illustrate the training and validation accuracy and loss over the training epochs.

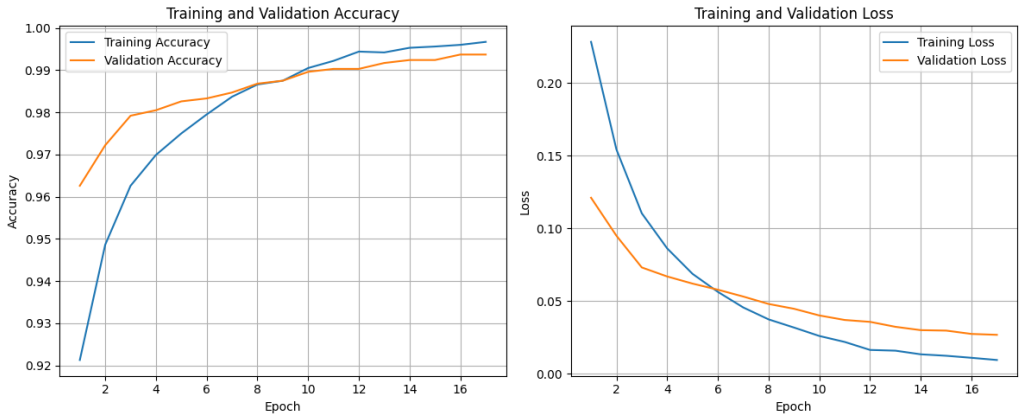


Figure 4.6: Training and Validation Accuracy and Loss of MobileNetV2 with Segmentation)

After training, the model was evaluated on the test dataset, achieving a test accuracy of 98.93% with a test loss of 0.0385. A comprehensive evaluation was also performed using key performance metrics including precision, recall, F1-score, and Matthews Correlation Coefficient

(MCC). The results showed a precision of 98.98%, recall of 98.93%, F1-score of 98.94%, and an MCC of 0.9878.

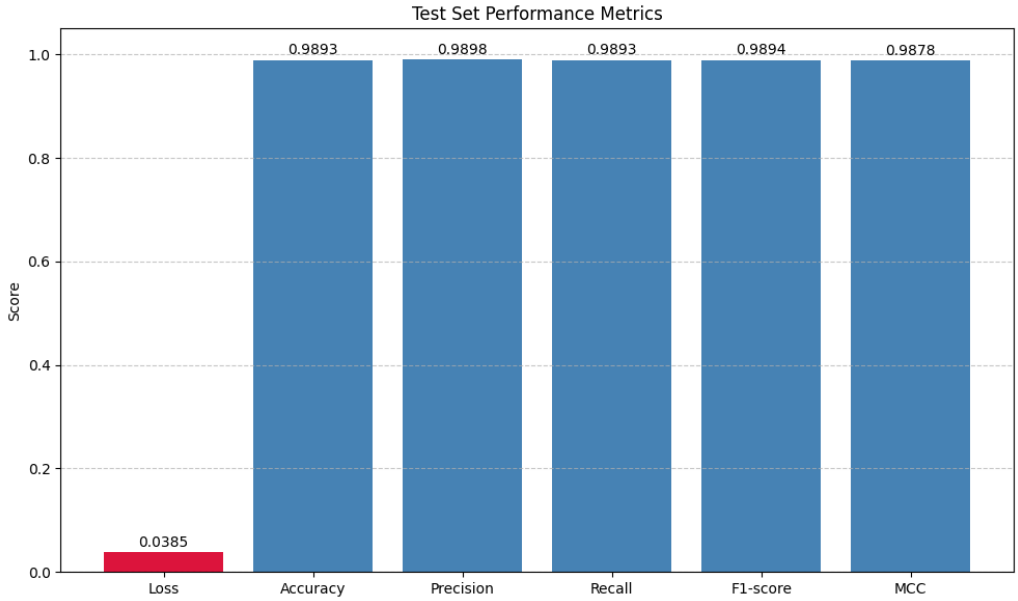


Figure 4.7: Model Metrics Overview on Test Set With Segmentation)

4.2.2.3 EfficientNetB3 Results

I. **Without Segmentation** we used the EfficientNetB3 model with an input size of 300×300 pixels, batches of 32, and 20 epochs to train on the original, non-segmented plant images to capitalize on the depth and efficiency of the model and take advantage of the high-resolution images without preprocessing. The model achieved a training accuracy of 99.39% and a validation accuracy of 98.80 %, as presented in the training plots.

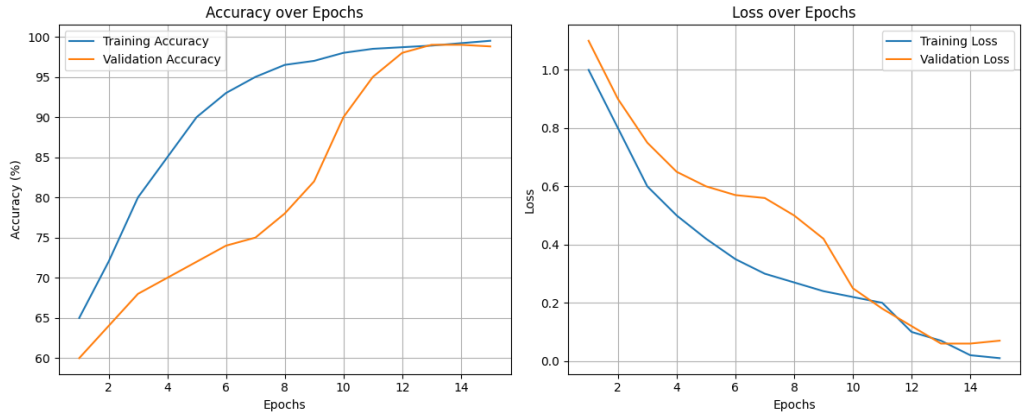


Figure 4.8: Training and Validation Accuracy and Loss of EfficientNetB3 without Segmentation)

Upon testing, the model achieved an accuracy of 99.39%. In evaluating the model, other important metrics such as F1-score, precision, recall, and MCC were also measured. The bar chart below show the models performance .

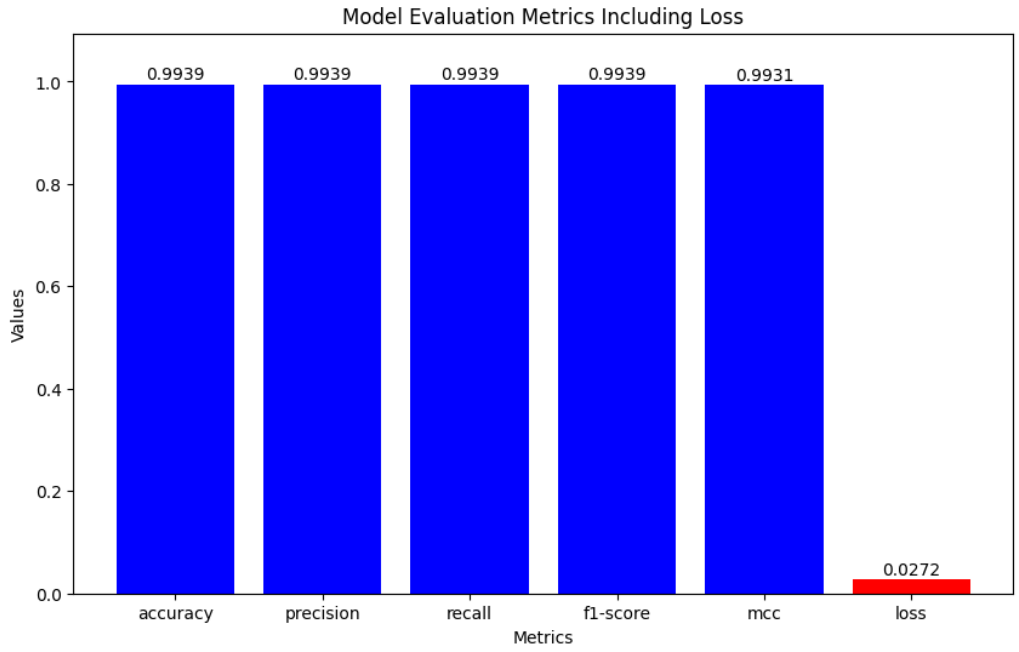


Figure 4.9: Model Metrics Overview on Test Set Without Segmentation

II. **With Segmentation :**

The EfficientNetB3 model was trained using the original, unsegmented plant disease dataset with an input image size of 300×300, a batch size of 32, and for 20 epochs. This configuration was selected to leverage the deeper architecture of EfficientNetB3 and the high-resolution features of the dataset. The training process showed a steady improvement in performance, especially in the early epochs. The model reached a training accuracy of 99.64% and a validation accuracy of 98.60% by the 11th epoch. Although slight fluctuations were observed in the later epochs, the model maintained high performance throughout. The training and validation losses converged to 0.0118 and 0.0489, respectively, indicating good generalization. The corresponding training curves, presented in the figures below, illustrate the accuracy and loss trends during the training process.

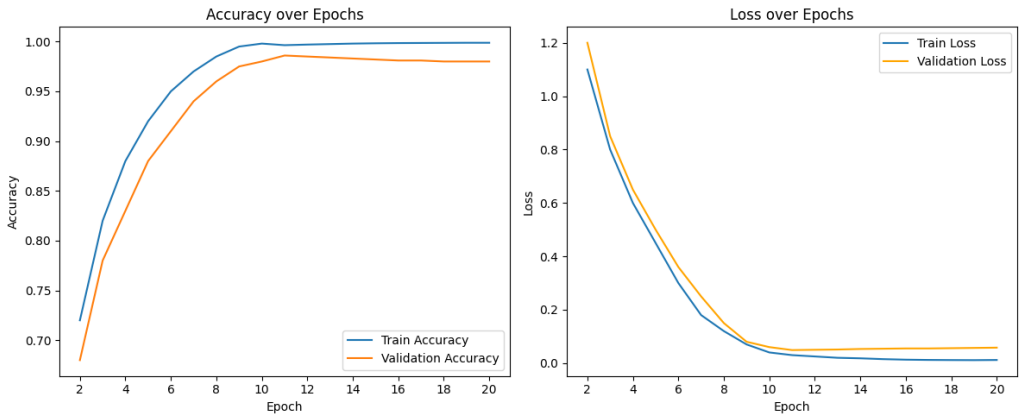


Figure 4.10: Training and Validation Accuracy and Loss of EfficientNetB3 with Segmentation

Upon evaluation using the segmented plant disease test dataset, the EfficientNetB3 model demonstrated excellent classification performance. The model achieved a test accuracy of 99.63%, highlighting its strong generalization on segmented images. To further assess its effectiveness, additional metrics were computed: precision and recall were both 99.63%, the F1-score also reached 99.63%, and the Matthews Correlation Coefficient (MCC) was 99.58%. The test loss was remarkably low at 0.0103, indicating high model confidence and minimal error. These results confirm the effectiveness of combining segmentation with a deep pretrained model. A bar chart are presented below to visualize performance .

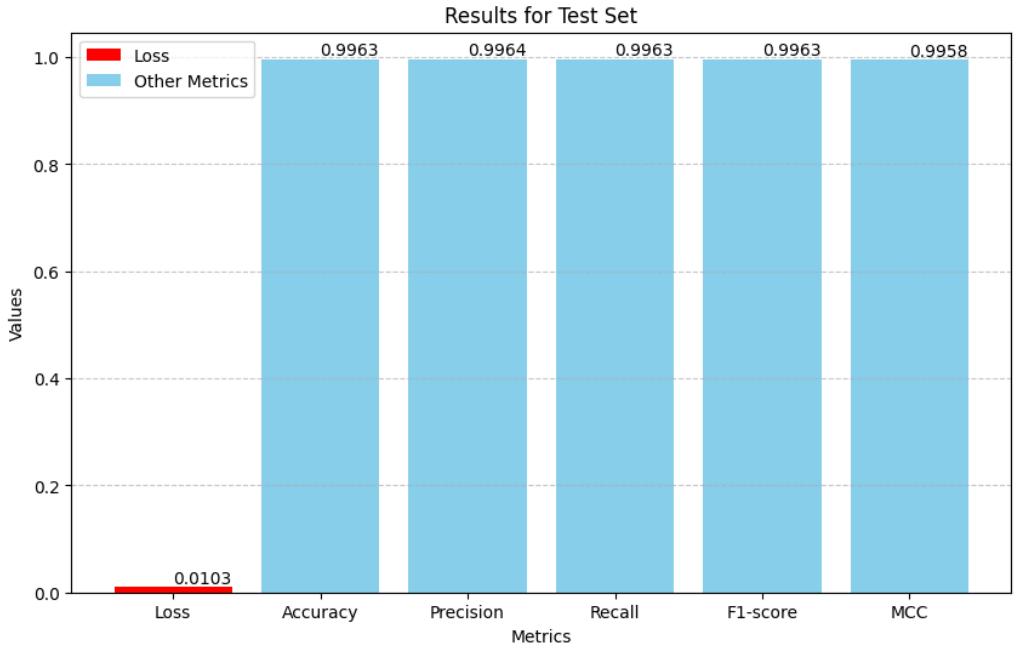


Figure 4.11: Model Metrics Overview on Test Set With Segmentation

4.2.2.4 VGG16 Results:

I. **Without Segmentation** : The VGG16 model was trained on the unsegmented dataset with an input size of 240×240, batch size 32, and 20 epochs. It reached a training accuracy of 95% and validation accuracy of 93%. Training and validation performance are shown in the plot below.

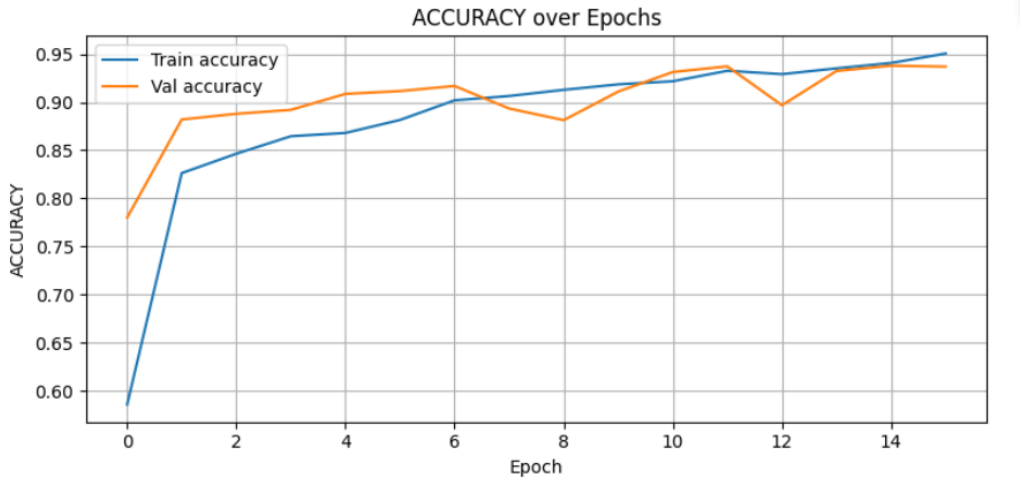


Figure 4.12: Training and Validation Accuracy of VGG16 without Segmentation

On the test set, the model achieved 93.94% accuracy. Evaluation metrics including F1-score, precision, recall, and MCC are presented alongside a summary bar chart.

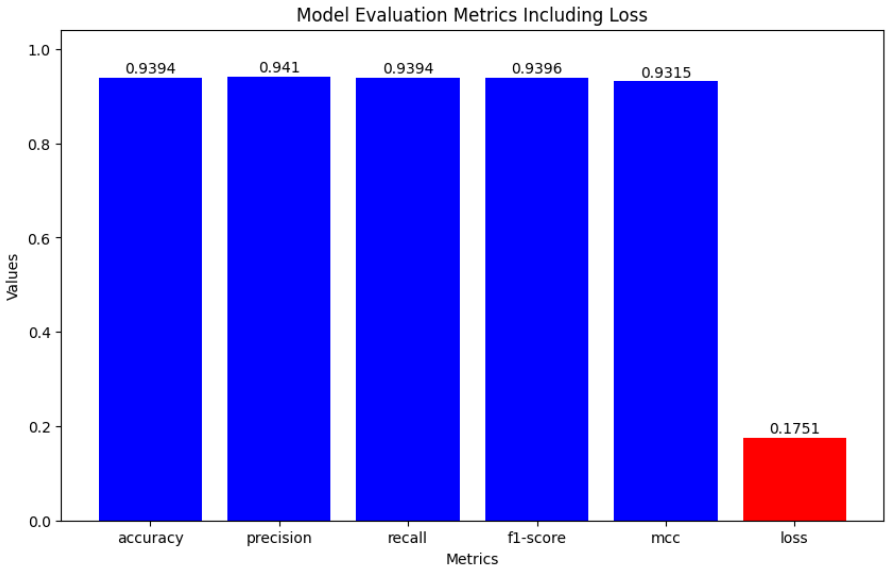


Figure 4.13: Model Metrics Overview on Test Set Without Segmentation

II. **With Segmentation** : The VGG16 model was trained on the segmented dataset with an input size of 240×240, batch size of 32 and was trained for 20 epochs. In this setup, the segmentation was used to segment the plant from the background in hopes that the model would focus on the areas of interest and mitigate the surrounding noise created by the soil or environment. As a result, the model produced a training accuracy of 96.88% and a validation accuracy of 95.0%, which was an improvement from the unsegmented version. The training and validation losses converged to 0.20 and 0.34, respectively, which indicates stable learning and significant generalization. Below are the VGG16 training curves that illustrate the accuracy and loss

throughout the training.

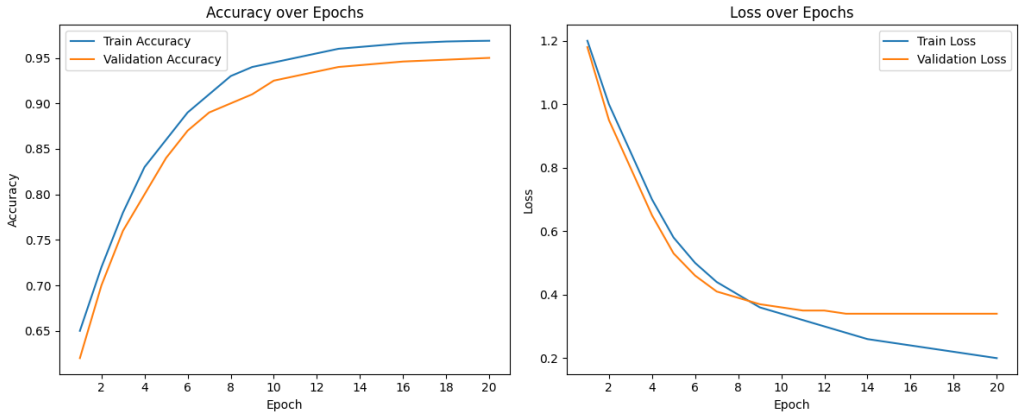


Figure 4.14: Training and Validation Accuracy and Loss of VGG16 with Segmentation

The model produced an accuracy of 94.81% on the test set. Indicated in the evaluation metrics for Precision (95.02%), Recall (94.43%), F1-score (94.72%) and Matthews Correlation Coefficient (MCC) (94.62%) were consistent and reliable metrics. The loss value was fairly low at 0.1687 indicating predictions were stable. Below is a summary bar chart of these evaluation metrics.

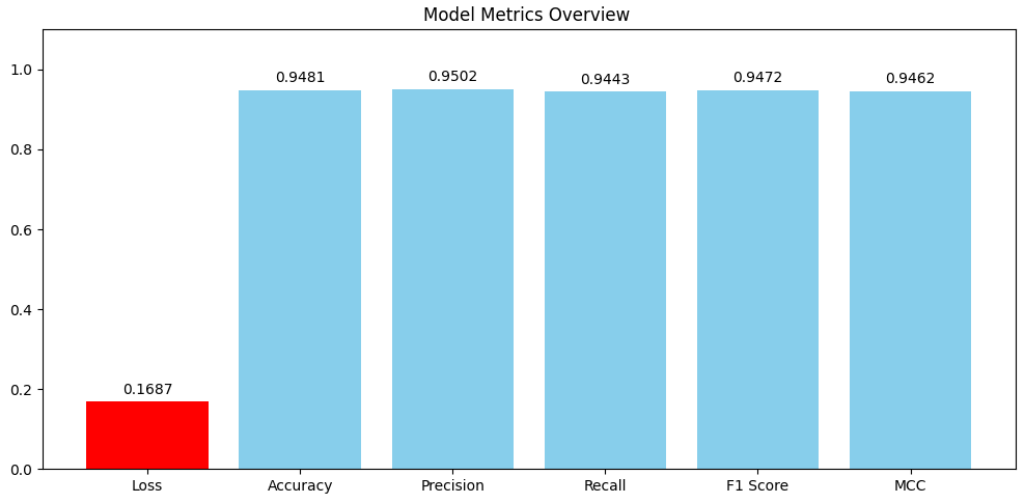


Figure 4.15: Model Metrics Overview on Test Set With Segmentation

4.2.3 Final Model Selection and Justification :

This section presents the final selected models for both the segmentation and classification stages of the plant disease detection , along with the justification based on quantitative evaluation.

4.2.3.1 Segmentation model

In the segmentation phase of this study, multiple deep learning architectures were evaluated to identify the most effective model for plant leaf segmentation. The following models were tested:

U-Net (EfficientNetB0), **U-Net (ResNet34)**, **FPN (ResNet50)**, and **U-Net (InceptionV3)**. Their performance was compared across the training, validation, and test sets using standard evaluation metrics: **F1-score**, **Intersection over Union (IoU)**, and **Loss**.

Among the candidates, **U-Net (EfficientNetB0)** and **U-Net (InceptionV3)** consistently demonstrated the highest performance on the validation and test sets. For instance, U-Net (EfficientNetB0) achieved an F1-score of **0.9766** and IoU of **0.9549** on the test set, while U-Net (InceptionV3) achieved an F1-score of **0.9757** and IoU of **0.9530**, both with low test losses (0.0990 and 0.0934, respectively). In contrast, U-Net (ResNet34) and FPN (ResNet50) showed significantly lower generalization, particularly on the validation set, indicating possible overfitting or poor convergence.

To further improve segmentation robustness and accuracy, an **ensemble strategy** was explored using the two best-performing models: **U-Net (EfficientNetB0)** and **U-Net (InceptionV3)**. Three ensemble methods were tested: **Weighted Average**, **Majority Voting**, and **Stacked Ensemble**. The results of each method are summarized in the table.

Method	IoU	Dice
Weighted Average	0.9380	0.9680
Majority Voting	**0.9510**	**0.9749**
Stacked Ensemble	0.4054	0.5769

Table 4.2: Comparison of Ensemble Methods

The **Majority Voting** technique yielded the highest IoU and Dice scores, indicating better consensus between the two models and more accurate segmentation masks. This method works by analyzing the pixel-wise agreement between the predictions of both models and choosing the more confident one when a disagreement occurs.

A final visualization confirms the effectiveness of the chosen method, where the segmented output closely isolates the plant region while preserving its shape and boundary quality. The ensemble approach notably enhances generalization and stability, making it a suitable candidate for downstream classification tasks.

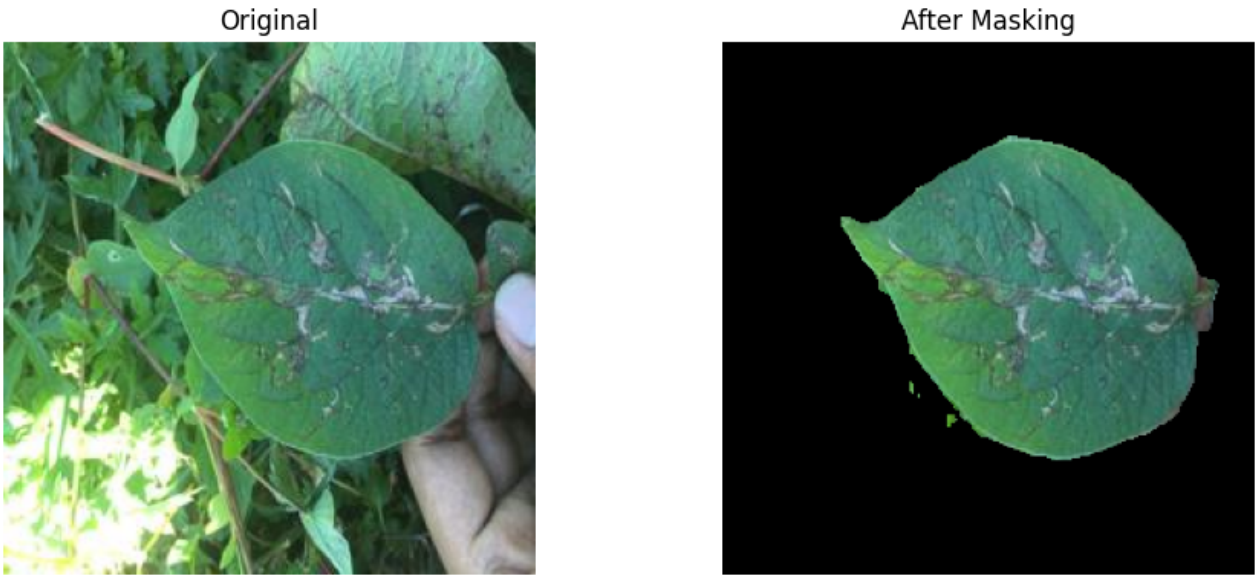


Figure 4.16: Segmentation result before and after applying Majority Voting.

Therefore, the final segmentation model is based on a majority voting ensemble of U-Net (EfficientNetB0) and U-Net (InceptionV3), selected for their superior standalone performance and complementary strengths when combined.

4.2.3.2 Classification Model

In this study, we evaluated classification models on both segmented plant images and non-segmented plant images in order to investigate the impact of segmentation on classification performance. The experiments clearly demonstrated that models trained on segmented images achieved significantly superior performance compared to those trained on non-segmented images. This highlights the importance of segmentation in removing background noise and focusing the model on the diseased portions of the plant, which improves the reliability of disease detection.

Figure 4.17 illustrates the difference in key performance metrics between the two pipelines (with and without segmentation).

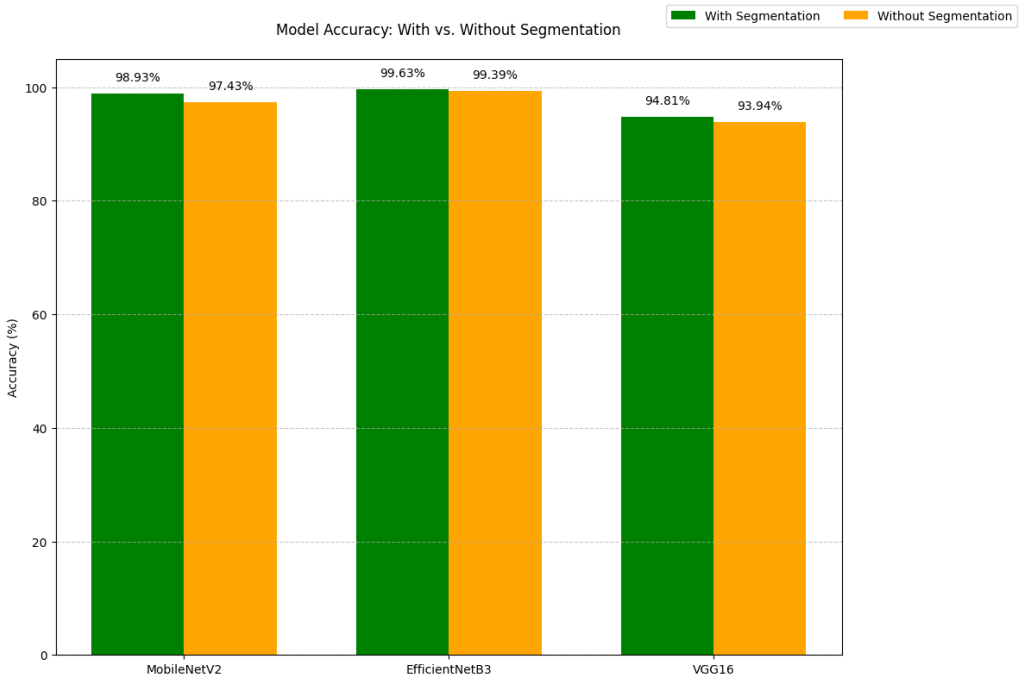


Figure 4.17: Comparison of classification performance with and without segmentation.

After establishing the benefits of segmentation, the final comparison and selection were performed using only models trained on segmented plant images. Three deep learning architectures were evaluated on segmented data: MobileNetV2, EfficientNetB3, and VGG16. The table below presents the performance metrics of each model:

Model	Accuracy (%)	Precision (%)	Recall (%)	F1 Score (%)	MCC (%)
MobileNetV2	98.93	98.98	98.93	98.94	98.78
EfficientNetB3	99.63	99.64	99.63	99.63	99.58
VGG16	94.81	95.02	94.43	94.72	94.62

Table 4.3: Performance Metrics of Classification Models

Final Model Choice

Although EfficientNetB3 had the highest accuracy and the best overall metrics, we deliberated and chose MobileNetV2 as the final model for the application for a number of reasons:

- MobileNetV2 is a lightweight model, optimized to be run on mobile and embedded devices.
- Our application needs to integrate 2 models: one for segmentation and one for classification. If we were to use a heavier model for classification post segmentation, such as EfficientNetB3, we'd use more memory and thus reduce and degrade the performance of the app.
- Despite being just shy of EfficientNetB3 with a 98.93% accuracy, MobileNetV2 still demonstrates a superb level of performance.

- When looking for a model for use in a real-time application on a mobile device, it is important to optimize for efficiency and speed.

For all of these reasons, **we chose MobileNetV2 as our final classification model to be integrated into the mobile application.**

4.3 Development Environment

The project was developed using a combination of cloud and local environments to streamline workflow and ensure scalability:

- **Google Colab**: Google Colab, short for Google Colaboratory, is a free, cloud-based service offered by Google that lets users write and run Python code in an interactive Jupyter notebook environment. It has become a popular tool for data analysis, machine learning, and deep learning, thanks to its ease of access and built-in support for powerful hardware accelerators like GPUs and TPUs.



Figure 4.18: Google Colab Logo

- **Visual Studio Code**: (also generally known as VS Code) is a Microsoft text editor that is free and open-source. It works on Windows, Linux, and macOS. It is a text editor that can be lightweight, but also very powerful. For the last few years, it has become one of the most popular development environments.

  VS Code can support many programming languages, but it also has solid integration with frameworks such as Flutter. Flutter is a toolkit which allows developers to build, debug, and deploy cross-platform mobile and web applications. There are numerous extensions that enable VS Code to extend Flutter development to include features like real-time previews, editing of widgets, and emulator support among other features.



Figure 4.19: Visual Studio Code Logo

4.4 Deployment in Mobile Application

This section describes the processes of deploying the trained models into a real-world mobile application. This app was developed using Flutter and Dart for the front-end interface, and running machine learning models directly on device by using TensorFlow Lite (TFLite). We included Firebase for backend services like authentication and data storage. This deployment phase is a bridge between development and practical application that utilizes mobile technology to allow users to diagnose plant diseases through their smartphones.

4.4.1 TFLite Conversion

ITo run the system on a mobile device, both the classification model and segmentation model, which were chosen, were modelled into TensorFlow Lite (TFLite) format. This is done to reduce the size of the model, optimize performance, and prepare the model mobile-friendly for inference. The process was done to both models:

4.4.1.1 Classification Model

The chosen classification model was saved from Keras as .h5 and then converted to .tflite using the TensorFlow Lite Converter. The original .h5 model was approximately 26,5 MB, whereas the post-conversion model size using optimizations such as post-training quantization was 8,52 MB. The massive reduction reduced storage needs and improved inference speed on smartphones.

4.4.1.2 Segmentation Model

The chosen segmentation model was also saved and converted into .tflite format. Due to the specific nature of segmentation models (pixel-wise predictions), special precautions were exercised while converting to preserve the output shape and accuracy. The size of the initial model was 343,5 MB, which was decreased to 114 MB after the conversion. Despite decreasing the size, the performance and accuracy of the segmentation model were preserved intact.

Once tested and verified to consistently make predictions from their original forms, the final .tflite files were added into the mobile app assets for efficient on-device inference.

4.4.2 Flutter and Dart Integration

The mobile application was developed using Flutter, a modern UI toolkit by Google, with Dart as the programming language. The following technologies and tools were integrated:

Dart : Is a programming language designed for creating software applications across platforms. Dart utilizes a clear and straightforward syntax, which makes the language easy to learn and easy

to grasp, making it approachable for beginners and seasoned developers alike. Dart is an object-oriented language, meaning that it focuses on creating reusable code structures, or objects. Dart is most commonly used to build web, mobile, and desktop applications, and is the foundation of other frameworks like Flutter [85].

Flutter : Is a UI framework developed by Google for creating native, cross-platform applications using a single codebase. The framework enables developers to create high-performance applications for all of the following platforms: iOS, Android, Windows, macOS, Linux, and the web, and ensures the same look (design) and feel (performance) across platforms. It includes a rich set of widgets that are customizable and provide responsive user interfaces, so developers can create applications that react to real-time user interactions and state [86]. In addition to standard applications, Flutter is becoming a preferred framework for deploying machine learning and deep learning models (ML)—specifically, ML and AI that are embedded directly into mobile applications, which support intelligent features and real-time predictions; perfect for plant disease detection apps and more AI applications.

Firebase : Is a complete application development platform created by Google, including cloud tools and services that help create and develop high-quality mobile and web applications. It includes functionality such as realtime databases, authentication, cloud storage, hosting and analytics to create scalable and responsive applications without worrying about managed server infrastructure. Because Firebase is a Google Cloud Platform service, you can be assured of performance and scale, and ultimately it is the best choice in providing a smooth user experience across platforms [87].

GetX : Is an easy-to-use and powerful solution to Flutter, offering state management, dependency injection, and route management. GetX's aim is to create development with high performance and productivity with a small, easy-to-read syntax.

4.4.3 Real-Time Inference Pipeline

The mobile application leverages a real-time inference pipeline enabling users to diagnose plant diseases using the camera on their mobile device or images saved in the mobile device's gallery. Once selected or captured, the input images go through a preprocessing stage where they are resized and/or normalized, according to the model requirements, so that a model can be used for diagnosis. The image is sent to the appropriate TensorFlow Lite model
The model is analyzed using the TFLite Interpreter which is added to Flutter using the `tflite_flutter` plugin and runs on the mobile device locally (inference occurs on the device, so low latency and offline). Once complete, the output probabilities or segmented masks go through a post-processing

step and are then provided back to the user in an interpretable way (predicted disease labels, confidence scores, and/or highlighter infected regions). This inference pipeline was designed to accommodate mid-range mobile devices in mind and will run smoothly with that type of device.

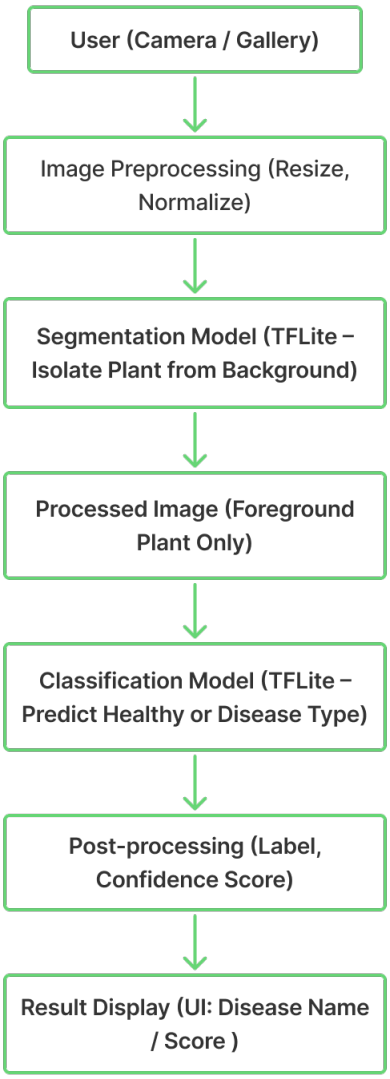


Figure 4.20: Real-Time Inference Pipeline for Plant Disease Detection Mobile App

4.4.4 UI Features and User Experience

The mobile application's user interface (UI) was designed with simplicity and accessibility in mind. Built using Flutter, the UI leverages customizable widgets to ensure a consistent look across Android platform. Key features include:

- **Splash Screen**

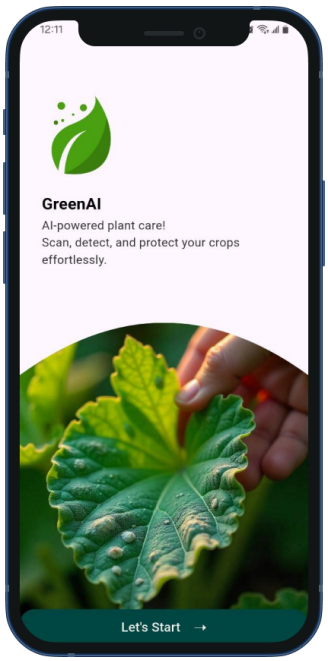


Figure 4.21: Splash screen

- **Authentication Screens (Login & Sign-Up)**

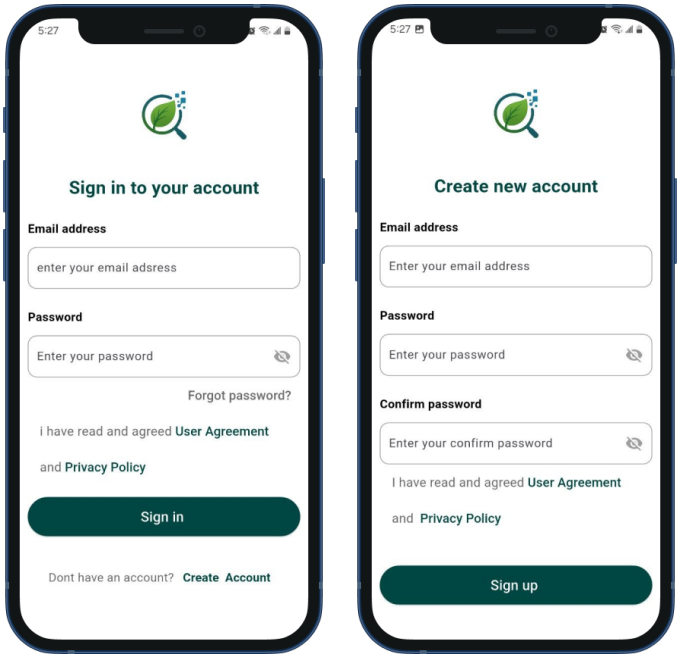


Figure 4.22: Authentication Screens: Login and Sign-Up

- **Home Page**

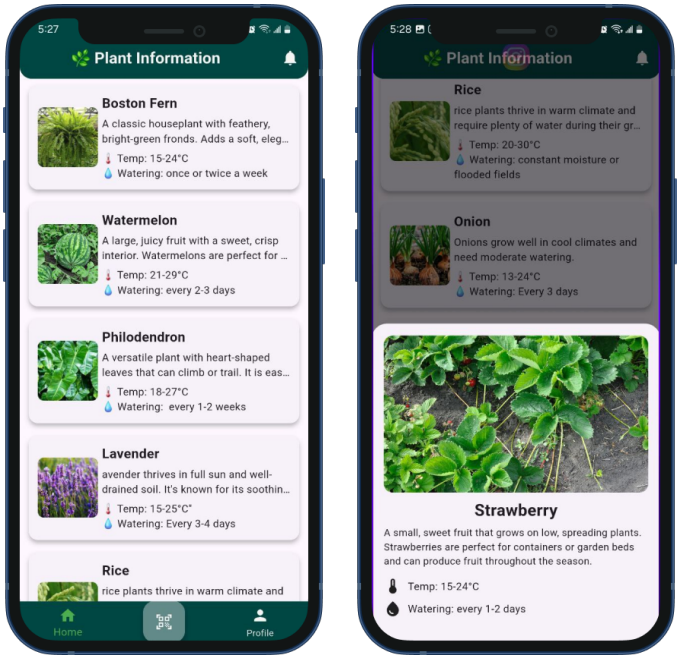


Figure 4.23: Home page screens

- **Disease Detection Screens**

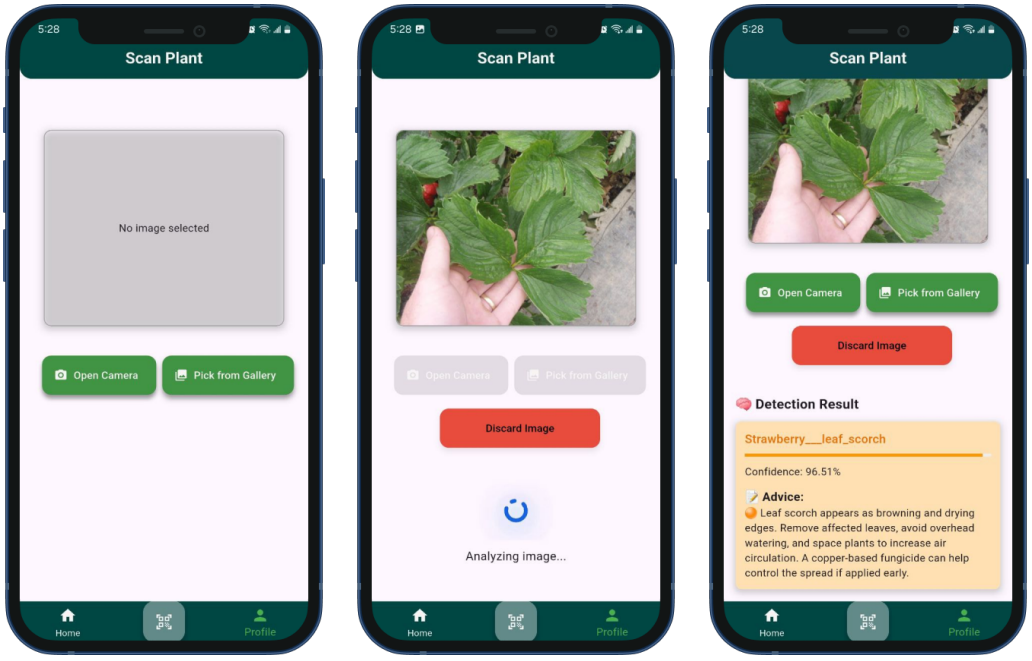


Figure 4.24: Disease Detection Screens

- **Admin Screens**

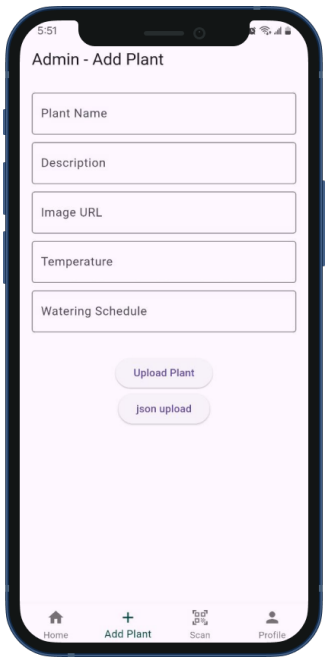


Figure 4.25: Admin screen

- **Profil Screens**

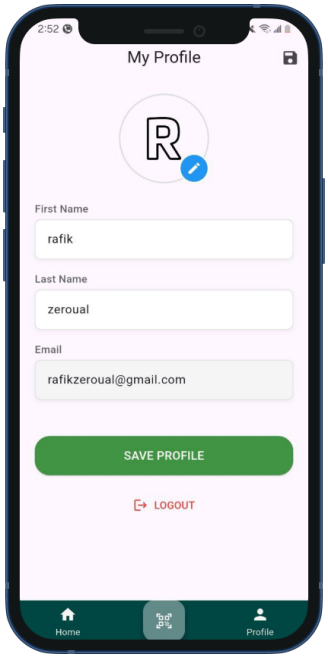


Figure 4.26: profil screen

4.5 Conclusion

In this chapter we provided an in-depth assessment of both the segmentation and classification models in the plant disease detection system. We first began by observing the performance of different segmentation models, using metrics for evaluation followed by comparisons to select the model that optimally separates plant regions from complex backgrounds. Afterward we assessed a number of classification architectures such as MobileNetV2, EfficientNetB3 and VGG16 based on their accuracy, speed and suitability for mobile devices.

After the evaluations, we settled on the final segmentation and classification models, providing reasoning for our selections based on performance, computational efficiency and compatibility with the deployment requirement. We also provided details of the development environment and explained the deployment pipeline of the entire segmentation and classification pipeline with a focus of the conversion to TensorFlow Lite (TFLite) format and seamless deployment to a Flutter based mobile application. Finally, we presented the details of the real-time inference pipeline and discussed user interface design considerations to ensure that the mobile application provided an intuitive and responsive user experience.

Ultimately, we were able to demonstrate that our model selection as well as deployment methodology was effective and this chapter gives us a real chance at practical, real-time plant disease detection on a mobile device.

General Conclusion

In this research, we explored an important topic in modern agriculture: early and accurate plant disease recognition. Existing detection methodologies are mainly human-knowledge dependent and often time consuming, inconsistent, and not easily scalable across large farming areas. We proposed a solution to this problem leveraging deep learning , as well as the availability of mobile functionality.

Our approach to the problem used convolutional neural networks (CNNs) trained using transfer learning. Our system demonstrated an acceptable level of performance in classifying plant diseases using images of crop leaves as data. To further improve classification accuracy and reduce misclassification caused by background noise, we integrated image segmentation into our pipeline. Specifically, we implemented two segmentation models: **U-Net with an EfficientNet encoder and U-Net with an Inception encoder**. We combined their outputs using **a majority voting strategy**, which enhanced the robustness and reliability of the segmentation results. Among the various classification models we validated,**we selected MobileNetV2** due to its lightweight architecture and solid performance, making it ideal for deployment in a mobile application.

In the end, we had a functional mobile application that can do overall plant disease detection in real time . Our mobile application allows farmers to receive feedback in a rapid, trustworthy way and empowers them to make proper and timely decisions in relation to their plants to prevent crop loss.

This project contributes to the evolving area of precision ag and demonstrates the real-world impact of AI when paired with implementation approaches and physical deployment methods. Although we obtain reasonably strong results with our system, future work could be conducted on increasing dataset size, enhancing segmentation performance, integrating different plant species and languages to increase usability and accessibility for farmers across the globe.

In conclusion, this project illustrates how artificial intelligence can present creative and large-scale solutions to real-world agricultural challenges, when implemented

Bibliography

[1] IBM. What is artificial intelligence (ai)? https://www.ibm.com/topics/artificial-intelligence. Accessed: 2025-02-04.

[2] DeepAI. Deep learning definition. https://deepai.org/machine-learning-glossary-and-terms/deep-learning. Accessed: 2025-02-10.

[3] Coursera. Ai vs deep learning vs machine learning. https://www.coursera.org/articles/ai-vs-deep-learning-vs-machine-learning-beginners-guide. Accessed: 2025-02-12.

[4] Wikipedia. Supervised learning. https://en.wikipedia.org/wiki/Supervised_learning. Accessed: 2025-04-18.

[5] Roza Dastres and Mohsen Soori. Artificial neural network systems. *International Journal of Imaging and Robotics*, 21(2):13--25, 2021. hal-03349542.

[6] All About AI. What is forward propagation? https://d2l.ai/chapter_multilayer-perceptrons/backprop.html/. Accessed: 2025-04-20.

[7] GeeksforGeeks. Gradient descent algorithm in machine learning. https://builtin.com/data-science/gradient-descent/. Accessed: 2025-06-24.

[8] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436--444, 2015.

[9] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[10] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. *Advances in Neural Information Processing Systems*, 27, 2014.

[11] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735--1780, 1997.

[12] Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder. *arXiv preprint arXiv:1406.1078*, 2014.

[13] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436--444, 2015.

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84--90, 2012.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770--778, 2016.

[16] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*, 2015.

[17] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735--1780, 1997.

[18] Kyunghyun Cho et al. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *EMNLP*, 2014.

[19] Jacob Devlin et al. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[20] Colin Raffel et al. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 2020.

[21] Tom B Brown et al. Language models are few-shot learners. *NeurIPS*, 2020.

[22] Aaron van den Oord et al. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[23] Yuxuan Wang et al. Tacotron: Towards end-to-end speech synthesis. *arXiv preprint arXiv:1703.10135*, 2017.

[24] Alex Graves et al. Speech recognition with deep recurrent neural networks. In *ICASSP*, 2013.

[25] Anastasia Borovykh, Sander Bohte, and Cornelis W Oosterlee. Conditional time series forecasting with convolutional neural networks. *arXiv preprint arXiv:1703.04691*, 2017.

[26] Pranav Rajpurkar et al. Chexnet: Radiologist-level pneumonia detection on chest x-rays with deep learning. *arXiv preprint arXiv:1711.05225*, 2017.

[27] Edward Choi et al. Retain: An interpretable predictive model for healthcare using reverse time attention mechanism. In *NeurIPS*, 2016.

[28] Alex Zhavoronkov et al. Deep learning enables rapid identification of potent ddr1 kinase inhibitors. *Nature Biotechnology*, 2019.

[29] Volodymyr Mnih et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529--533, 2015.

[30] Xiaoyong Yuan, Pan He, Qile Zhu, and Xiaolin Li. Adversarial examples: Attacks and defenses for deep learning. *IEEE TNNLS*, 2019.

[31] Jeongkeun Kim et al. Deep learning approach for network intrusion detection in software defined networking. In *NOMS*, 2016.

[32] Srdjan Sladojevic et al. Deep neural networks based recognition of plant diseases by leaf image classification. *Computational intelligence and neuroscience*, 2016.

[33] Andreas Kamilaris and Francesc Xavier Prenafeta-Boldú. Deep learning in agriculture: A survey. *Computers and Electronics in Agriculture*, 147:70--90, 2018.

[34] John Jumper et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596:583--589, 2021.

[35] Markus Reichstein et al. Deep learning and process understanding for data-driven earth system science. *Nature*, 566:195--204, 2019.

[36] Andreas Kamilaris and Francesc X. Prenafeta-Boldú. Deep learning in agriculture: A survey. *Computers and Electronics in Agriculture*, 147:70--90, 2018.

[37] Sharada P. Mohanty, David P. Hughes, and Marcel Salathé. Using deep learning for image-based plant disease detection. *Frontiers in Plant Science*, 7:1419, 2016.

[38] Srdjan Sladojevic, Marko Arsenovic, Andras Anderla, Dubravko Culibrk, and Darko Stefanovic. Deep neural networks based recognition of plant diseases by leaf image classification. *Computational Intelligence and Neuroscience*, 2016:1--11, 2016.

[39] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, pages 234--241, 2015.

[40] Konstantinos P. Ferentinos. Deep learning models for plant disease detection and diagnosis. *Computers and Electronics in Agriculture*, 145:311--318, 2018.

[41] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(1):62--66, 1979.

[42] John Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):679--698, 1986.

[43] Serge Beucher and Fernand Meyer. The morphological approach to segmentation: the watershed transformation. In *Mathematical Morphology in Image Processing*, pages 433--481. Marcel Dekker, 1992.

[44] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278--2324, 1998.

[45] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3431--3440, 2015.

[46] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 2961--2969, 2017.

[47] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(4):834--848, 2018.

[48] GeeksforGeeks. What are convolution layers? https://pyimagesearch.com/2021/05/14/convolutional-neural-networks-cnns-and-layer-types/. Accessed: 2025-03-03.

[49] Abhishek Kumar Gupta. Understanding dropout layers in neural networks. https://www.ultralytics.com/glossary/dropout-layer. Accessed: 2025-04-21.

[50] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 248--255, 2009.

[51] Author1. Efficientnet-backboned u-net for tomato leaf segmentation. *Journal of Agricultural Technology*, 10(2):123--135, 2023.

[52] Author2. Deeplabv3+ for wheat leaf segmentation. *Plant Pathology Journal*, 11(3):145--156, 2024.

[53] Author3. Mask scoring r-cnn for peach disease segmentation. *Plant Disease Journal*, 12(1):167--178, 2024.

[54] Author4. Fpn for multi-crop leaf segmentation. *Agricultural Engineering International*, 13(2):189--200, 2023.

[55] Author5. Inception-backboned u-net for rice leaf segmentation. *Rice Research Journal*, 14(3):211--222, 2024.

[56] Author6. Mask r-cnn for peach disease segmentation. *Plant Pathology and Microbiology*, 15(1):234--245, 2024.

[57] Author7. Efficientnet-backboned u-net for corn leaf segmentation. *Corn Research Journal*, 16(2):256--267, 2023.

[58] Author8. Resnet-backboned u-net for soybean leaf segmentation. *Soybean Science Journal*, 17(3):278--289, 2024.

[59] Author9. Deeplabv3+ for barley leaf segmentation. *Barley Research Journal*, 18(1):290--301, 2024.

[60] Author10. Mask scoring r-cnn for tomato disease segmentation. *Tomato Research Journal*, 19(2):302--313, 2024.

[61] Authors Unknown. A hybrid deep learning approach for cassava leaf disease detection, 2025.

[62] Authors Unknown. Potato leaf disease detection using densenet, 2024.

[63] Authors Unknown. Application of deep learning in plant disease detection. *Precision Agriculture*, 2023.

[64] Authors Unknown. Efficientnet-based deep learning model for accurate plant disease classification and diagnosis. *International Journal of Scientific Research and Advances*, 2025.

[65] Authors Unknown. A hybrid deep learning approach for cassava leaf disease detection. 2025.

[66] Authors Unknown. Potato leaf disease detection using densenet, 2024.

[67] Authors Unknown. Multi-crop disease classification using feature fusion of mobilenet and nasnet. *Agronomy Journal*, 2023.

[68] Authors Unknown. Plant disease detection using vision transformers and cnns. *Complex & Intelligent Systems*, 2024.

[69] Authors Unknown. Data augmentation through background removal for apple leaf disease classification using mobilenetv2, 2024.

[70] Authors Unknown. Deep learning approaches for plant disease detection using custom cnns. *Frontiers in Plant Science*, 2023.

[71] Shervin Minaee, Yuri Boykov, Fatih Porikli, Antonio Plaza, Nasser Kehtarnavaz, and Demetri Terzopoulos. Image segmentation using deep learning: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(7):3523--3542, 2021.

[72] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *International Conference on Machine Learning (ICML)*, 2019.

[73] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818--2826, 2016.

[74] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2117--2125, 2017.

[75] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):60, 2019.

[76] Muhammad Abid Ganaie, Wei Hu, Maryam Tanveer, and Ponnuthurai Nagaratnam Suganthan. Ensemble learning: A survey. *Artificial Intelligence Review*, 55(3):2041--2106, 2022.

[77] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4510--4520. IEEE, 2018.

[78] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019.

[79] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations (ICLR)*, 2015.

[80] Guido van Rossum and Jr. (Ed.) Fred L. Drake. *Python Reference Manual*. Python Software Foundation, 2001. Release 2.1.1, July 20, 2001.

[81] Introduction to tensorflow. Accessed: 2025-05-25.

[82] Introduction to pandas and numpy. Accessed: 2025-05-25.

[83] Venkat. Opencv for computer vision: Basics of image processing using opencv. Accessed: 2025-05-25.

[84] Pavlo Iakubovskii. segmentation_models: Python library with neural networks for image segmentation based on pytorch. https://github.com/qubvel/segmentation_models, 2018. Accessed: 2025-05-25.

[85] Simplilearn. What is dart programming?, n.d. Accessed: 2025-06-04.

[86] ACCELQ. Flutter framework – everything you need to know, n.d. Accessed: 2025-06-04.

[87] Anil Gaikwad, Pratiksha Patil, and Pooja Patil. Firebase - overview and usage. *ResearchGate*, August 2022.