B.Tech. BCSE497J - Project-I

DEEP LEARNING-BASED DETECTION OF PLANT LEAF DISEASE USING REAL-WORLD AGRICULTURAL DATASETS

Submitted in partial fulfillment of the requirements for the degree of

Bachelor of Technology

in

Computer Science and Engineering

by

22BCE2694 MULLAPUDI YASWANTH SRI SAI VENKAT

22BCE2348 AVULA VENKAT SREERAM CHANDRA

22BCE3821 MUDUNURI SAI RANGA RAMA PAVAN KUMAR RAJU

Under the Supervision of

Dr. K. Ragavan

Assistant Professor Senior Grade I

School of Computer Science and Engineering (SCOPE)

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

November 2025

DECLARATION

We hereby declare that the project entitled **Deep Learning-Based Detection Of Plant Leaf Disease Using Real-World Agricultural Datasets** submitted by us, for the award of the degree of *Bachelor of Technology in Computer Science and Engineering* to VIT is a record of bonafide work carried out by us under the supervision of Prof. / Dr. **RAGAVAN K .**

We further declare that the work reported in this project has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place : Vellore

Date : 12-11-2025

[illegible]

[illegible]

[illegible]

Signature of the Candidate

CERTIFICATE

This is to certify that the project entitled **Deep Learning-Based Detection Of Plant Leaf Disease Using Real-World Agricultural Datasets** submitted by Mullapudi Yaswanth Sri Sai Venkat(22BCE2694) Avula Venkata Sreeram Chandra(22BCE2348) Mudunuri Sai Ranga Rama Pavan Kumar Raju(22BCE3821) School of Computer Science and Engineering, VIT, for the award of the degree of *Bachelor of Technology in Computer Science and Engineering*, is a record of bonafide work carried out by Us under my supervision during Fall Semester 2025-2026, as per the VIT code of academic and research ethics.

The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university. The project fulfills the requirements and regulations of the University and in my opinion meets the necessary standards for submission.

Place : Vellore

Date : 12-11-2025

VELLORE INSTITUTE OF TECHNOLOGY (VIT) [illegible] School of Computer Science & Engineering (SCOPE) Vellore-632 014, T.N., India

[illegible] 12/11/2025
Signature of the Guide

[illegible] 12/11/2025
[illegible] 12/11/2025
Examiner(s)

Dr. BOOMINATHAN P

Head Of The Department Software Systems

School Of Computer Science and Engineering

ACKNOWLEDGEMENTS

We are deeply grateful to the management of Vellore Institute of Technology (VIT) for providing us with the opportunity and resources to undertake this project. Their commitment to fostering a conducive learning environment has been instrumental in our academic journey. The support and infrastructure provided by VIT have enabled us to explore and develop my ideas to their fullest potential.

We sincerely thanks to Dr. Jaisankar N, the Dean of the School of Computer Science and Engineering (SCOPE), for his unwavering support and encouragement. His leadership and vision have greatly inspired us to strive for excellence. The Dean's dedication to academic excellence and innovation has been a constant source of motivation for us. We appreciate his efforts in creating an environment that nurtures creativity and critical thinking.

We express Our profound appreciation to Dr. BOOMINATHAN P, the Head of the SCHOOL OF COMPUTER SCIENCE AND ENGINEERING (SOFTWARE SYSTEMS), for his/her insightful guidance and continuous support. His/her expertise and advice have been crucial in shaping the direction of our project. The Head of Department's commitment to fostering a collaborative and supportive atmosphere has greatly enhanced our learning experience. His constructive feedback and encouragement have been invaluable in overcoming challenges and achieving my project goals.

We Are immensely thankful to our project guide Dr. RAGAVAN K, for his dedicated mentorship and invaluable feedback. His patience, knowledge, and encouragement have been pivotal in the successful completion of this project. our supervisor's willingness to share his/her expertise and provide thoughtful guidance has been instrumental in refining our ideas and methodologies. His support has not only contributed to the success of this project but has also enriched our overall academic experience.

Thank you all for your contributions and support.

Mudunuti Sai Ranga Durga Pavan Kumar Raju
M. Yaswanth Sri Sai [illegible]
A. Venkata Sreeram Chandra Vardhan.

Name of the Candidate

TABLE OF CONTENTS

List of Figures

List of Tables

List of Abbreviations

Abbreviation	Full form	Short description
RGB	Red-Green-Blue	Color image channels.
IoT	Internet of Things	Edge devices / sensors for field deployment.
GPU	Graphics Processing Unit	Hardware accelerator for model training/inference.
SSIM	Structural Similarity Index	Perceptual image similarity metric (0–1).
MSE	Mean Squared Error	Pixel-wise reconstruction loss.
IoU	Intersection over Union	(if used) segmentation / overlap metric.
LR	Learning Rate	Optimizer step-size hyperparameter.
Adam	Adaptive Moment Estimation	Optimizer used for training.
TFLite	TensorFlow Lite	Mobile/edge model format.

Symbols and Notations

Symbol / Notation	Description
(K)	Number of classes (disease categories)
(Yi)	True one-hot label for class (i)
(TP)	True positives for class
(FP)	False positives for class
(FN)	False negatives for class
(TN)	True negatives for class
(T)	Total number of training epochs
(t)	Current epoch (training iteration)
PSNR	Peak Signal-to-Noise Ratio (image quality metric)
SSIM	Structural Similarity Index (image similarity metric)
MSE	Mean Squared Error (reconstruction/pixel error)

ABSTRACT

Early detection of plant leaf diseases is vital for protecting crops from severe yield losses and ensuring food security. Conventional manual inspection methods are time-consuming, error-prone, and require expert supervision, making them unsuitable for large-scale agricultural monitoring. Deep learning-based computer vision approaches have recently emerged as a reliable solution for automated plant disease identification. However, many existing models still struggle to maintain an effective balance between classification accuracy, generalization ability, and computational efficiency, especially under real-world field conditions.

A previous hybrid architecture combining AlexNet and ShuffleNet achieved nearly 95% classification accuracy on controlled datasets but exhibited limited robustness and higher dependency on computational resources. These shortcomings make such models impractical for mobile or IoT-based agricultural environments, where power and memory are constrained. Moreover, factors such as variable lighting, occlusion, and background noise in real-world images further challenge the performance of these traditional architectures.

To overcome these limitations, this study introduces a Hybrid EfficientNetB0–MobileNetV2 deep learning framework designed for accurate and efficient plant leaf disease detection. The proposed model combines the compound-scaling capability of EfficientNetB0, which ensures rich hierarchical feature extraction, with the lightweight and resource-efficient structure of MobileNetV2. Together, they form a robust architecture capable of high-performance classification while maintaining a small model footprint suitable for edge deployment. A comprehensive preprocessing pipeline—including noise reduction, image normalization, and extensive data augmentation—was employed to enhance feature diversity and reduce overfitting. Transfer learning and fine-tuning techniques were used to further optimize the hybrid model on real-world agricultural datasets.

The system was evaluated using four distinct datasets comprising rice, sugarcane, blackgram, and tomato leaves, covering both healthy and multiple diseased classes. Performance metrics including accuracy, precision, recall, and specificity were computed to assess model effectiveness. Experimental results demonstrated significant improvements over the baseline AlexNet–ShuffleNet hybrid model, achieving higher accuracy, improved sensitivity to subtle disease symptoms, and better generalization across unseen data. The results confirm that the hybrid architecture successfully balances predictive performance with computational efficiency.

CHAPTER 1

1. INTRODUCTION

1.1 Background

Agriculture serves as the fundamental pillar of global food security and economic stability. However, this critical sector is under immense pressure from climate volatility, increasing soil heterogeneity, and the growing demand for food from a rising global population. Traditional farming methods, often reliant on historical precedent and generational knowledge, are increasingly challenged by these modern complexities. An incorrect crop selection can lead to depleted soil nutrients, wasted resources, and significant economic losses for farmers.

This project focuses on the development of an efficient and accurate system for plant leaf disease detection using deep learning techniques. The proposed work is inspired by the research paper *"Deep Learning-Based Detection of Plant Leaf Disease Using Real-World Agricultural Datasets" (2025)*, which utilizes a hybrid AlexNet–ShuffleNet model. To improve performance while maintaining computational efficiency, this study replaces the hybrid architecture with EfficientNetB0, a state-of-the-art convolutional neural network known for its balance between accuracy and resource usage. Additionally, MobileNetV2 is explored to further enhance lightweight feature extraction capabilities.

The primary objective of this project is to design a model that achieves high classification accuracy with low computational cost, making it suitable for real-time agricultural applications such as mobile-based disease monitoring systems. Through the integration of EfficientNetB0, the system aims to outperform previous architectures in both efficiency and reliability, contributing to sustainable and technology-driven farming practices.

1.2 Motivations

Plant diseases are one of the leading causes of reduced crop productivity and economic losses in agriculture. Farmers often depend on manual visual inspection or expert consultation to identify diseases, which can be inefficient, time-consuming, and prone to human error. With the advancement of deep learning and image processing technologies, there is an increasing opportunity to automate disease detection through computer vision systems. However, many existing deep learning architectures require high computational power, making them impractical for deployment on mobile or edge devices in rural areas.

Key Motivations:

- To reduce dependency on manual and expert-based disease diagnosis methods.
- To minimize crop losses by enabling early and accurate disease detection.
- To utilize deep learning for automating plant health monitoring.

- To design a model that performs efficiently on mobile and low-power devices.
- To support the adoption of smart and sustainable farming practices.
- To contribute to food security through technology-driven agricultural innovation.
- To improve accuracy and generalization using the EfficientNetB0+MoblieNetV2 architecture.

1.3 Scope of the Project

The scope of this project encompasses the development, training, and evaluation of a deep learning-based model for automated plant leaf disease detection using real-world agricultural datasets. The project focuses on leveraging the EfficientNetB0+MoblieNetV2 architecture, known for its balance between accuracy and computational efficiency, to build a system capable of recognizing multiple leaf diseases with high precision.

This work not only aims to improve classification accuracy but also to ensure that the solution is lightweight, scalable. The project further explores data preprocessing, augmentation techniques, and comparative analysis with other models like EfficientNetB0 and MobileNetV2 to validate performance improvements.

Key Areas Covered within the Project Scope:

- Development of a deep learning model using EfficientNetB0 for disease classification.
- Use of a real-world leaf datasets with different classes of diseases and healthy samples.
- Implementation of image preprocessing and augmentation to improve model generalization.
- Comparative performance evaluation with other models (e.g., AlexNet, ShuffleNet, MobileNetV2).
- Model training, validation, and testing using preprocessed and augmented datasets.
- Analysis of performance metrics such as accuracy, precision, recall ,specificity and confusion matrix.
- Contribution toward precision agriculture and smart farming systems.

CHAPTER 2

2. PROJECT DESCRIPTION AND GOALS

2.1 Literature Review

Recent advancements in deep learning have significantly enhanced the accuracy and efficiency of plant disease detection. Mohanty et al. [3] were among the first to demonstrate the potential of deep convolutional neural networks (CNNs) for identifying plant leaf diseases from image data. Their model successfully classified multiple plant species, highlighting the feasibility of using transfer learning for agricultural applications.

Hosny et al. [5] introduced a hybrid model that fused CNN features with Local Binary Pattern (LBP) descriptors to improve texture representation in plant leaf images. This method was tested on tomato, grape, and apple datasets and achieved high classification accuracy, showcasing the advantages of combining handcrafted and deep features. Similarly, Vishnoi et al. proposed a CNN[6] model for detecting apple leaf diseases such as scab, cedar rust, and black rot using the PlantVillage dataset. Their lightweight architecture reduced computational overhead and was suitable for real-time applications in handheld or mobile devices.

Shovon et al. [4] developed PlantDet, a deep ensemble framework based on InceptionResNetV2, Xception, and EfficientNetV2L models. By integrating regularization, advanced activation functions, and visualization tools like Grad-CAM, the model achieved robust and interpretable results across multiple plant datasets. In a related study, Sunil et al. [2] employed EfficientNetV2 for cardamom and grape disease detection, demonstrating improved feature extraction and model generalization using multiscale processing and transfer learning.

Further enhancements have been observed in lightweight architectures. Zhang et al. [7] introduced ShuffleNet, a highly efficient CNN optimized for mobile devices, which inspired later research in portable plant disease detection systems.
While existing studies have achieved strong classification performance, most struggle to balance accuracy, generalization, and computational efficiency. To overcome these limitations, the present work proposes a hybrid EfficientNetB0–MobileNetV2 architecture, combining the compound-scaled feature extraction power of EfficientNetB0 with the lightweight design of MobileNetV2. This integration enables high-accuracy detection while maintaining low computational cost, making it well-suited for real-world agricultural and mobile deployment scenarios.

2.2 Research Gap

From the literature review on existing plant leaf disease detection methods, the following key gaps have been identified:

1. **Limited feature extraction capability** – Many existing models, such as simple AlexNet-based architectures, fail to capture deep spatial and texture-level information required to distinguish between visually similar leaf diseases.
2. **Low generalization across datasets** – Several studies achieved high accuracy only on small or controlled datasets, but their performance significantly dropped when tested on real-world agricultural images with varying lighting, background, and noise conditions.
3. **Insufficient model optimization** – Prior models often rely on a single deep network, leading to either underfitting (lightweight models) or overfitting (heavy models) without balanced accuracy and efficiency.
4. **Lack of hybrid and knowledge distillation approaches** – Few existing studies utilize feature fusion or knowledge transfer techniques to combine the strengths of multiple architectures for improved performance and reduced computational load.
5. **High computational complexity** – Many transformer-based architectures provide good accuracy but require high-end hardware, making them unsuitable for real-time field use by farmers and agricultural experts.

These gaps highlight the need for a lightweight, hybrid deep learning framework that combines EfficientNetB0 and MobileNetV2 for enhanced feature extraction, better generalization, and computational efficiency, while also providing accurate disease classification with confidence-based predictions suitable for real-time agricultural applications.

2.3 Objectives

The primary goal of this project is to design and implement an efficient deep learning model capable of accurately detecting and classifying different types of tomato leaf diseases using real-world agricultural datasets. To achieve this, the project combines EfficientNetB0 and MobileNetV2 architectures into a hybrid framework to balance accuracy, computational efficiency, and deployment feasibility for smart agricultural systems.

Specific Objectives:

1. To develop a hybrid deep learning architecture by combining EfficientNetB0 and MobileNetV2 for accurate and lightweight plant leaf disease detection.
2. To preprocess and augment real-world tomato leaf images to enhance model generalization and reduce overfitting under variable environmental conditions.
3. To train and validate the hybrid model using a different dataset containing six classes of leaf images in different leaf images like (*Bacterial Spot, Black Mold, Gray Spot, Healthy, Late Blight,* and *Powdery Mildew*).

4. To compare the hybrid model's efficiency against existing architectures (e.g., AlexNet–ShuffleNet, EfficientNetB0, MobileNetV2) in terms of accuracy and computational cost.
5. To contribute to sustainable agriculture by reducing dependency on manual inspection and enabling early disease detection to minimize crop losses.

2.4 Problem Statement

Plant diseases are a major threat to global food security, significantly reducing crop yield and quality every year. In traditional farming practices, disease identification is often done manually by farmers or agricultural experts through visual inspection, which is time-consuming, subjective, and prone to human error. While deep learning techniques have shown great promise in automating plant disease detection, existing models such as AlexNet, VGGNet, and ResNet are computationally intensive, making them unsuitable for real-time or mobile-based deployment in agricultural settings.

Although several lightweight architectures like ShuffleNet and MobileNetV2 have been proposed, they often compromise accuracy for speed, limiting their reliability in field environments. Similarly, hybrid models such as AlexNet–ShuffleNet improved efficiency but still struggled with the balance between performance and resource utilization, particularly when dealing with real-world datasets that include variations in lighting, background, and leaf orientation.

Therefore, there is a pressing need to develop a hybrid deep learning model that achieves both high accuracy and low computational complexity, enabling robust disease detection even under real-world conditions. This project addresses this challenge by designing a hybrid EfficientNetB0–MobileNetV2 model that leverages EfficientNetB0's strong feature extraction and MobileNetV2's computational efficiency to create a balanced and scalable system for leaf disease classification.

2.5 Project Plan

Gantt Chart Description: Plant Leaf Disease Detection Project (July – November 2025)

The Gantt chart illustrates the complete project workflow for the **Plant Leaf Disease Detection System** carried out between **July and November 2025**. The project is organized into sequential phases, beginning from planning and literature review to model design, training, testing, optimization, and final report submission. Each phase contributes to the systematic development of a deep learning-based hybrid model for accurate disease detection in plant leaves.

Phase 1 – Project Planning and Problem Identification (July 1 – July 10, 2025)

The project begins with identifying the core research problem — the limitations of existing plant disease detection systems that fail to achieve high accuracy, generalization, and computational efficiency. During this phase, the objectives are finalized, and the scope of the proposed **EfficientNetB0 + MobileNetV2 hybrid model** is clearly defined to ensure a lightweight, accurate, and real-time solution.

Phase 2 – Literature Review and Gap Analysis (July 11 – July 20, 2025)

An in-depth literature review is conducted to study traditional AlexNet, ShuffleNet, and other hybrid deep learning models used for agricultural disease detection. Based on the analysis, key research gaps are identified, such as limited feature extraction, high computational cost, and lack of hybrid learning strategies. These findings justify the need for a more efficient and edge-optimized hybrid model.

Phase 3 – Dataset Collection and Pre-processing (July 21 – August 5, 2025)

In this phase, datasets containing Blackgram leaf images (both healthy and diseased) are collected from real-world agricultural sources. The data is divided into training (80%) and testing (20%) sets. Image preprocessing includes resizing to 224×224 pixels, normalization, and filtering to remove noise. Augmentation techniques such as rotation, flipping, zooming, and brightness adjustment are applied to enhance the model's robustness.

Phase 4 – Extraction and Hybrid Model Design (August 6 – August 25, 2025)

The images to focus on disease-affected leaf areas. A hybrid deep learning architecture combining EfficientNetB0 and MobileNetV2 is designed. Both networks are loaded with pre-trained ImageNet weights, and the final layers are fine-tuned for feature fusion. This hybrid model aims to combine EfficientNetB0's high feature richness with MobileNetV2's lightweight performance.

Phase 5 – Model Training and Validation (August 26 – September 15, 2025)

The hybrid model is trained using the processed datasets with categorical cross-entropy loss and Adam optimizer. A cosine annealing learning rate scheduler and label smoothing ($\varepsilon = 0.1$) are applied to stabilize training. EarlyStopping and ModelCheckpoint callbacks ensure the best version of the model is saved. Both training and validation accuracy are monitored for 100 epochs to achieve optimal performance.

Phase 6 – Model Testing and Result Evaluation (September 16 – October 5, 2025)

The trained model is evaluated on the unseen test dataset to assess its predictive performance. Evaluation metrics such as accuracy, precision, recall, and specificity are calculated. The confusion matrix is plotted to visualize correct and incorrect classifications. Additionally, visual testing is conducted where the model predicts whether a leaf is healthy or unhealthy, and if unhealthy, it identifies the specific disease type along with confidence value.

Phase 7 – Result Analysis and Hyperparameter Tuning (October 6 – October 25, 2025)

The results are analyzed to fine-tune the model's hyperparameters for further improvement. Adjustments to learning rate, batch size, and dropout values are made to enhance generalization. Comparative analysis is performed between the proposed hybrid model and traditional models (e.g., AlexNet–ShuffleNet) to demonstrate accuracy improvement and efficiency gain.

Phase 8 – Documentation and Final Report Submission (October 26 – November 11, 2025)

In the final phase, the entire project process, experimental results, and graphical outputs are compiled into the final documentation and project report. Visual results, confusion matrices, and performance graphs are included for clarity. The completed report and model files are reviewed, finalized, and submitted before November 11, 2025.

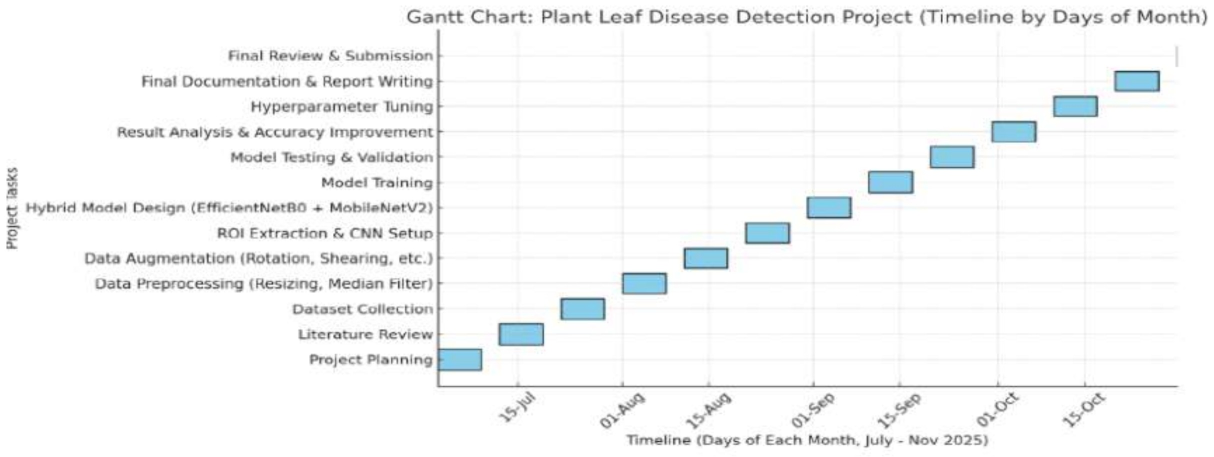


Fig 2.5: Gnatt Chart

CHAPTER 3

3. TECHNICAL SPECIFICATION

3.1 Requirements

3.1.1 Functional Requirements

1. **Dataset Management**
   The system shall load, organize, and manage image datasets of rice, sugarcane, blackgram, and tomato leaves containing both healthy and diseased samples for training and validation.
2. **Data Preprocessing and Augmentation**
   The system shall preprocess all images by resizing, normalizing, and cleaning them or uniformity.It shall also perform data augmentation techniques such as rotation, flipping, and contrast adjustment to increase dataset size and improve generalization.
3. **Hybrid Model Development**
   The system shall construct a hybrid deep learning model by combining EfficientNetB0 and MobileNetV2 architectures for enhanced feature extraction and classification performance.

4. **Model Training and Optimization**
   The system shall train the hybrid model using the prepared datasets and optimize hyperparameters such as learning rate, batch size, and epochs to achieve higher accuracy and lower validation loss.
5. **Performance Evaluation and Result Generation**
   The system shall evaluate model performance using metrics such as accuracy, precision, recall and specificity and display the final results to measure model effectiveness and improvement.
6. **Model Saving and Testing**
   The system shall save the trained model and weights for future use and test the model on unseen leaf images to validate its generalization capability.

3.1.2 Non Functional Requirements

1. **Performance**
   The system shall provide high accuracy in disease classification while maintaining efficient training and inference times.It should utilize GPU resources effectively to speed up model training and minimize computation time.
2. **Scalability**
   The system shall be scalable to include additional crops, disease types, or larger datasets without significant loss in performance or efficiency.
3. **Reliability**
   The system shall ensure consistent and repeatable results across multiple training runs with the same dataset and parameters.It should handle data or process interruptions gracefully without crashing.
4. **Usability**
   The system shall have a clear, modular structure allowing researchers or developers to easily modify code, update models, and retrain with new data.Proper documentation shall be provided for smooth usage and experimentation.
5. **Efficiency**
   The system shall use computational resources efficiently by leveraging batch processing, optimized data pipelines, and memory management during training.
6. **Accuracy and Robustness**
   The system shall achieve high classification accuracy across multiple crops and be robust to variations in lighting, background, and image quality.

3.2 Feasibility Study

3.2.1 Technical Feasibility

The proposed system is technically feasible as it utilizes well-established deep learning frameworks such as TensorFlow and Keras to build and train the hybrid EfficientNetB0 +

MobileNetV2 model. These frameworks provide built-in functions for image preprocessing, data augmentation, and model optimization, reducing implementation complexity. The required dataset of leaf images for rice, sugarcane, blackgram, and tomato crops is available and can be efficiently processed using GPUs to accelerate training and improve accuracy.

The system's architecture is modular, making it easy to integrate, modify, and extend with newer versions of models or additional crop datasets. The use of pre-trained models enables efficient transfer learning, which significantly reduces computational requirements and allows implementation even on standard hardware with GPU support. Therefore, the system is technically achievable with currently available tools and technology.

3.2.2 Economic Feasibility

The system is economically feasible as it primarily depends on open-source software tools and frameworks such as Python, TensorFlow, Keras, and OpenCV, eliminating the need for expensive licenses. The datasets used for training and validation are publicly available, further reducing costs. Since the model training and testing can be performed using standard computing resources or cloud-based GPU services, no significant hardware investment is required.

Once developed, the system can be deployed at minimal cost, making it accessible to educational institutions, research organizations, and farmers. The return on investment is high because it helps in early detection of diseases, preventing crop losses and improving agricultural productivity. Thus, the system offers long-term cost-effectiveness compared to manual monitoring or external consultancy services.

3.2.3 Operational Feasibility

Operationally, the proposed system is easy to use, maintain, and extend. Since it is a backend-based model without a complex user interface, it can be operated by researchers, students, or agricultural experts familiar with Python-based environments. The system's workflow—loading datasets, training models, and viewing output results—is straightforward and well-documented.

The trained hybrid model can be easily updated with new datasets to adapt to additional crops or diseases, ensuring continuous improvement. The system's operational requirements such as basic computational resources, storage, and GPU support are manageable for most research or academic setups. Therefore, the project is operationally feasible for long-term use and adaptation.

3.3 System Specification

The system specification defines the minimum hardware and software configuration required for the successful development and execution of Plant leaf disease datasets using the Hybrid model Proper specifications ensure smooth model training, testing, and result analysis.

3.1 Hardware Specification

Component	**Specification**
Processor (CPU)	Intel Core i7 or higher / AMD Ryzen 7 or higher
Graphics Processing Unit (GPU)	NVIDIA GeForce RTX 3060 or higher (for faster computation)
RAM	Minimum 16 GB (Recommended: 32 GB for large datasets)
Storage	500 GB Solid-State Drive (SSD) for faster data access
Power Supply Unit (PSU)	750 W or higher to support GPU and other components
Cooling System	High-performance air cooling or liquid cooling system
Display	Full HD (1920 × 1080) resolution or higher
Network Connectivity	Stable internet connection (for dataset download and library updates)

Table 3.1: Hardware Specification

Description:
The hardware setup focuses on providing high computational speed and stability during training and testing of deep learning models. A GPU with multiple cores and higher memory bandwidth ensures faster training, while sufficient RAM and SSD storage provide smooth data handling and quick model loading.

3.2 Software Specification

Software	**Description**
Operating System	Windows 10 / Windows 11 / Ubuntu Linux
Programming Language	Python 3.8 or higher
Deep Learning Frameworks	TensorFlow, Keras, or PyTorch
Image Processing Libraries	OpenCV, PIL (Python Imaging Library), scikit-image
Supporting Libraries	NumPy, Pandas, Matplotlib, Seaborn
Development Environment	Google Colab/ Jupyter Notebook
Dataset Used	Plant Leaf Disease (base paper)
Kaggle	Leaf images

Table 3.2: Software Specification

Description:
The software configuration provides a complete environment for deep learning model development. Python acts as the core programming language, while frameworks such as TensorFlow and Keras . Libraries like OpenCV and scikit-image assist in image preprocessing and augmentation.Development environments such as Google Colab are recommended as they offer free GPU access for faster computation and easy collaboration.

CHAPTER 4

4. Design Approach and Details

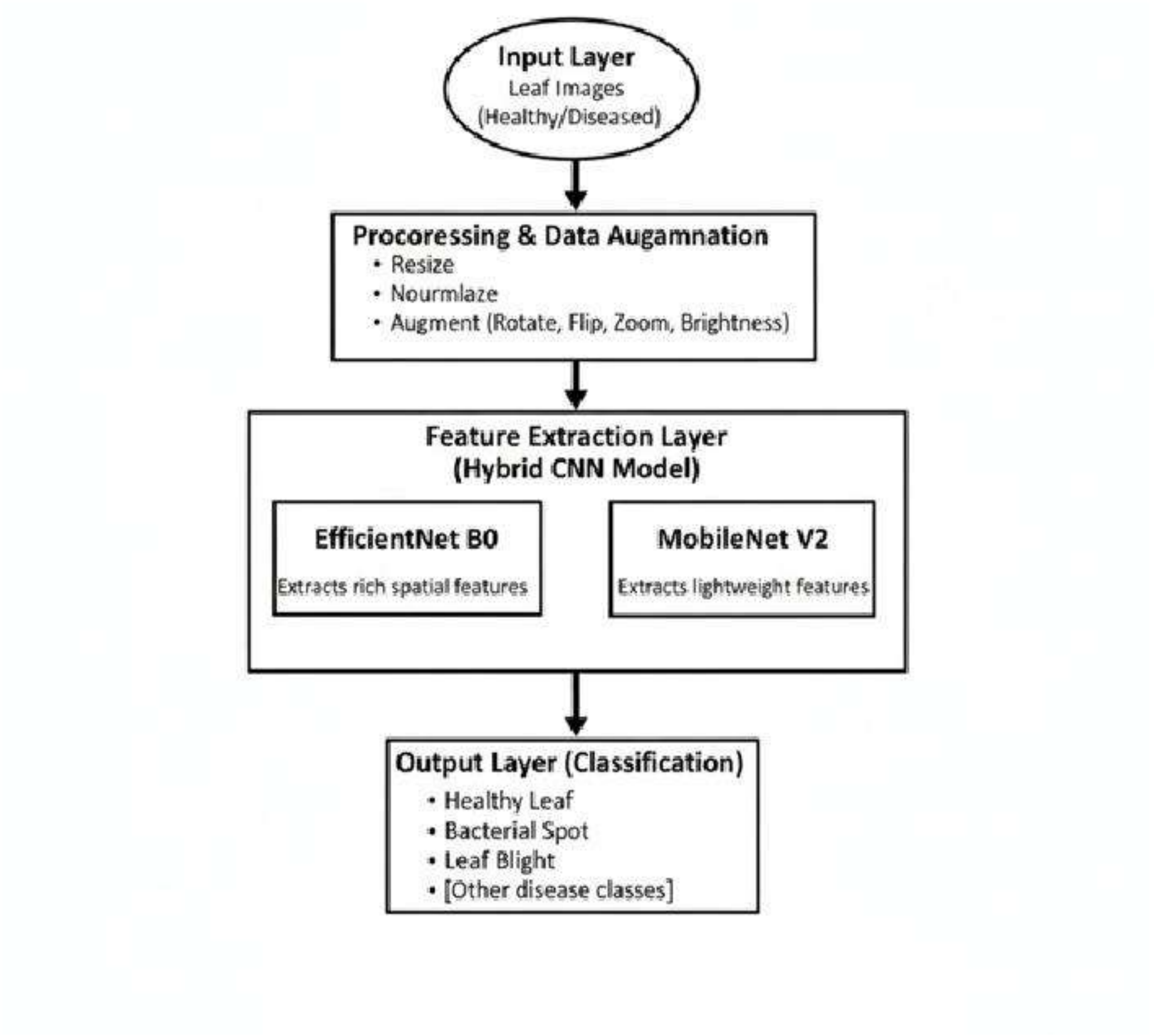
This chapter explains the overall design and workflow of the Plant Leaf Disease . It focuses on how the system is structured, how data flows through different components, and how each module interacts with others to achieve the final output.

4.1 System Architecture

The system follows a modular, dataset-driven deep learning architecture optimized for efficient training, feature extraction, and accurate classification of plant leaf diseases across multiple crops such as rice, sugarcane, blackgram, and tomato.

Architecture Description:

1. **Input Layer:**
   Leaf images from the dataset are provided as input to the system. These images belong to different categories such as healthy or diseased (various disease types per crop).
2. **Preprocessing Layer:**
   All input images are resized to a uniform dimension, normalized to a standard pixel scale, and augmented (rotation, flipping, zooming, brightness adjustment) to increase dataset diversity and improve model generalization.
3. **Feature Extraction (Hybrid Layers):**
   The hybrid model combines EfficientNetB0 and MobileNetV2 architectures. Both networks extract deep spatial features like textures, spots, and colour variations from leaf images, which are important indicators of disease patterns.
4. **Feature Fusion Layer:**
   The extracted features from both models are concatenated (fused) to form a single rich feature vector, combining EfficientNetB0's high-level semantic details and MobileNetV2's lightweight feature efficiency.
5. **Fully Connected (Dense) Layers:**
   The fused feature vector is passed through fully connected layers with activation and dropout functions to perform nonlinear mapping and reduce overfitting.
6. **Output Layer:**
   The final layer classifies the input image into one of the defined categories — such as Healthy Leaf, Bacterial Spot, Leaf Blight, or other disease classes — depending on the crop type.



Input Layer
Leaf Images
(Healthy/Diseased)
Procoressing & Data Augamnation
• Resize
• Nourmlaze
• Augment (Rotate, Flip, Zoom, Brightness)
Feature Extraction Layer
(Hybrid CNN Model)
EfficientNet B0
Extracts rich spatial features
MobileNet V2
Extracts lightweight features
Output Layer (Classification)
• Healthy Leaf
• Bacterial Spot
• Leaf Blight
• [Other disease classes]

Fig 4.1: System Architecture

4.2 Design

4.1 Block Diagram

The below diagram illustrates the complete workflow of the *Plant Leaf Disease Detection System* using a hybrid deep learning model. The architecture represents the end-to-end process starting from the input image to the final disease classification output. It highlights how image preprocessing, augmentation, and hybrid model fusion contribute to accurate plant health prediction.

1. Input Image Acquisition

The system begins with the input image, which represents a leaf sample collected from crops such as rice, sugarcane, blackgram, and tomato. These images may be captured under varying environmental conditions using mobile cameras or field sensors. The input dataset contains both healthy and diseased leaf samples, forming the foundation for model training and testing.

2. Preprocessing

Once the raw image is collected, it passes through the preprocessing stage, where operations like image resizing and median filtering are applied.

- **Image resizing** ensures all input images are converted to a uniform dimension suitable for the neural network input size.
- **Median filtering** helps reduce background noise and smoothens image textures without losing important leaf features.

3. Data Augmentation

In this stage, data augmentation is applied to increase dataset diversity and improve model generalization. Various transformations are performed, such as:

- Rotation – to simulate different leaf orientations,
- Shearing, Zooming and Flipping – to imitate different scales and angles or inverted leaf images
- Brightness adjustment – to mimic different lighting conditions. This process produces multiple variations of each image, helping the model learn to recognize diseases under real-world variability.

5. Hybrid Deep Learning Model

The preprocessed and augmented images are then fed into the hybrid deep learning architecture, which integrates two models:

- **EfficientNetB0** : Responsible for high-level, fine-grained feature extraction using compound scaling techniques that balance depth, width, and resolution.
- **MobileNetV2**: A lightweight network that captures essential spatial patterns with fewer parameters and faster computation.
  Both models process the input images in parallel, extract complementary features, and fuse them at the feature vector level. This feature fusion enhances overall detection accuracy while maintaining computational efficiency.

6. Classification Layer

The combined features are then passed through fully connected dense layers, which classify the images into one of two categories:

- **Healthy** — indicating a disease-free leaf, or
- **Unhealthy** — indicating the presence of a disease.
  If the image is classified as *unhealthy*, the model proceeds to further categorize the type of disease.

7. Output and Disease Identification

Finally, the system outputs the classification result along with a confidence score, showing how certain the model is about its prediction. The output distinguishes between healthy and diseased leaves, and for diseased samples, specifies the type of disease detected.

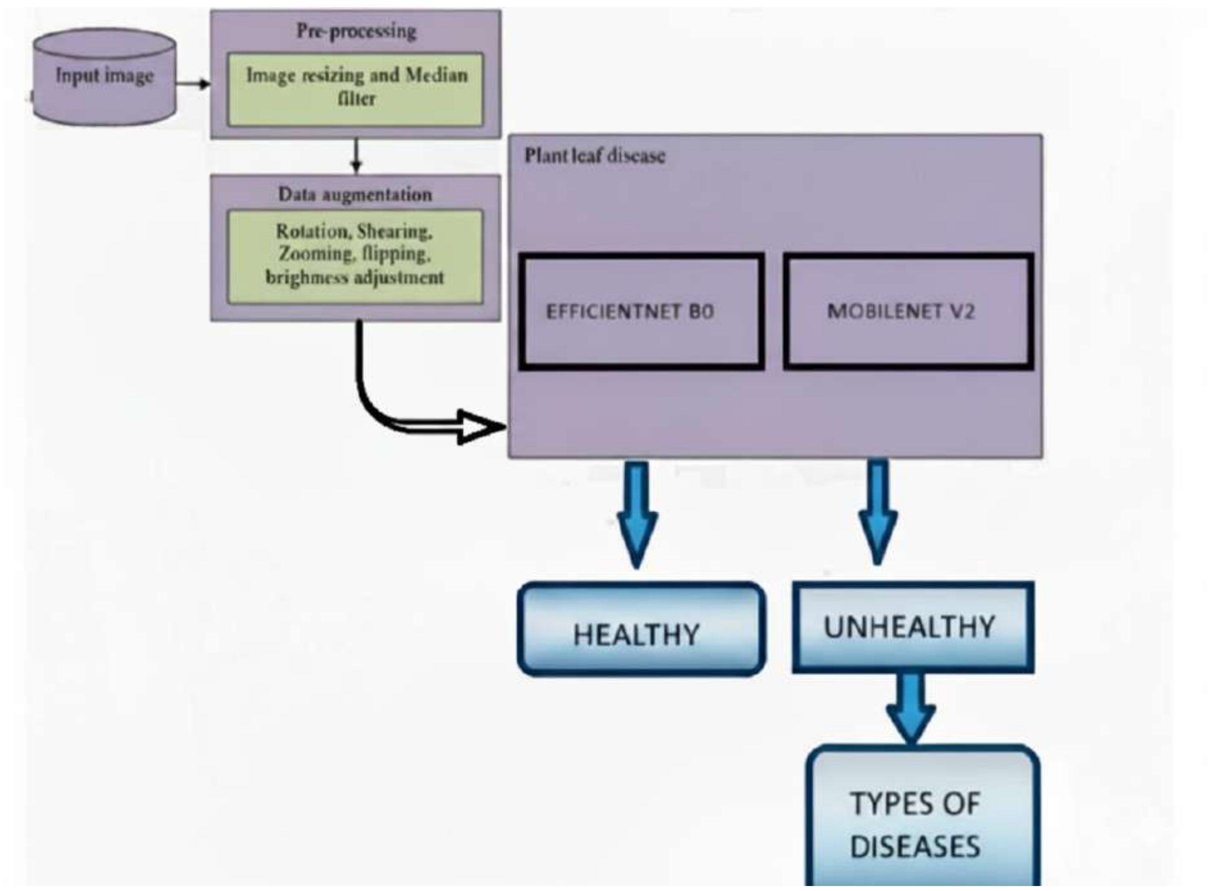


Fig 4.1: Block Diagram

4.2 Class Diagram

1. DatasetLoader Class

The DatasetLoader class is responsible for loading and validating image datasets for different crops such as rice, sugarcane, blackgram, and tomato.
It defines attributes like:

- data_path: The directory path where the datasets are stored.
- split_ratios: The percentage distribution for training, validation, and testing sets.
- load(): Reads the dataset from the given directory and organizes it into appropriate data structures.
- validate(): Ensures that all images are correctly labeled and checks for missing or corrupted files.

2. Preprocessor Class

The Preprocessor class performs image preprocessing operations to prepare raw images for model input.It standardizes image dimensions and ensures that every sample has consistent quality.

Attributes:

- Image_size: The target size (e.g., 224×224) required for EfficientNetB0 and MobileNetV2.

Functions:

- resize(img): Resizes the input image to the defined size.
- normalize(img): Normalizes pixel intensity values to a 0–1 range.
- denoise(img): Removes background noise or blur using filters.
- transform(dataset): Applies all preprocessing steps sequentially to the dataset.

3. Augmentor Class

The Augmentor class enhances the dataset by creating synthetic variations of existing images through transformations.It helps improve the model's robustness to lighting, orientation, and environmental variations.

Attributes: (policies)- A list of augmentation techniques such as rotation, flipping, brightness change, and zoom.

Functions:

- augment(img): Applies a random augmentation policy to a single image.
- apply(dataset): Applies augmentations to the entire dataset to increase its size and diversity.

4. ModelBuilder Class

The ModelBuilder class constructs the hybrid architecture by combining EfficientNetB0 and MobileNetV2.

Attributes:

- EfficientNetB0_config: Configuration parameters for EfficientNetB0, such as weights and input shape.
- mobilenet_config: Configuration for MobileNetV2, including depth multiplier and activation settings.

Functions:

- build_hybrid(input_shape): Creates both EfficientNetB0 and MobileNetV2 models, fuses their feature layers, and builds the combined network.

- compile(model, params): Compiles the model with optimizer, loss function, and evaluation metrics.

5. Trainer Class

The Trainer class manages the training process of the hybrid model.It controls the learning process, monitors performance, and tunes hyperparameters for better accuracy.

Attributes:

- optimizer: Defines the optimization algorithm (e.g., Adam or RMSProp).
- callbacks: A list of Keras callbacks such as EarlyStopping, ReduceLROnPlateau, and ModelCheckpoint.

Functions:

- train(model, train_ds, val_ds): Trains the model using training and validation datasets while recording accuracy and loss.
- fine_tune(model, layers): Unfreezes selected layers for fine-tuning after initial training.

6. Evaluator Class

The Evaluator class handles the testing and performance analysis of the trained hybrid model.

Functions:

- evaluate(model, test_ds): Computes metrics such as accuracy, precision, recall, and Specificity using the test dataset.
- confusion_matrix(y_true, y_pred): Generates a confusion matrix to visualize correct vs. incorrect predictions.
- generate_reports(metrics): Summarizes the results and produces a report for performance interpretation.

7. ModelSaver Class

The ModelSaver class manages the saving and loading of trained models for reuse or deployment.

Functions:

- save(model, path): Saves the trained model to a specified directory in .h5 or .keras format.
- load(path): Loads a previously saved model for evaluation or inference.
- export_tflite(model, path): Converts and exports the model into a lightweight TensorFlow Lite format suitable for mobile or IoT deployment.

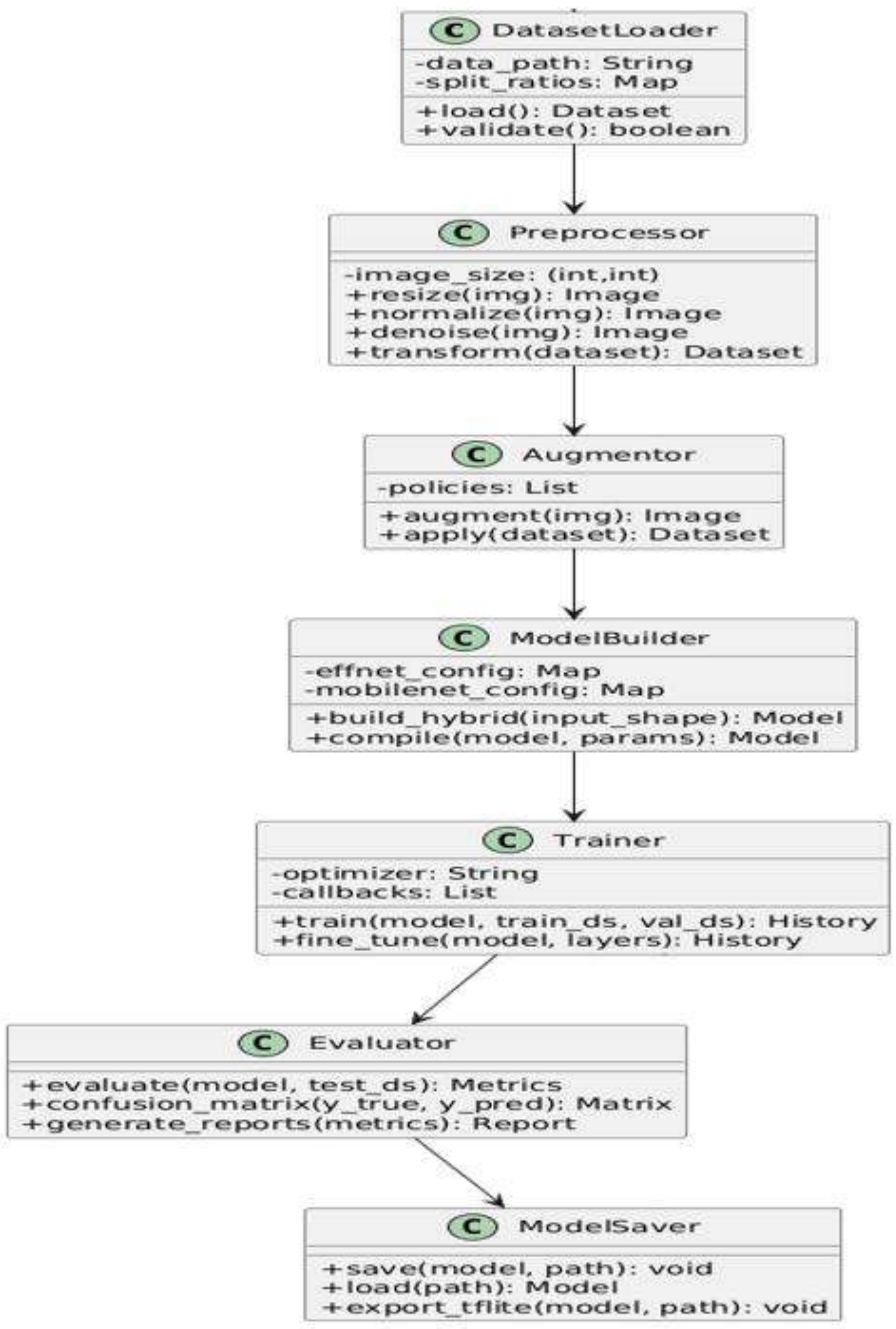


Fig 4.2: Class Diagram

4.3 Use Case Diagram

The Use Case Diagram for the Plant Leaf Disease Detection System illustrates the interaction between the Researcher/User and the system's core functionalities. The main actor in this system is the Researcher/User, who performs all operations related to data handling, model training, and disease prediction.

The process begins when the user loads the dataset of plant leaf images into the system. The user then proceeds to preprocess the images by resizing and normalizing them, followed by data augmentation to enhance the dataset with variations such as rotation, flipping, and brightness adjustments.

Once the data is ready, the user trains the hybrid model (EfficientNetB0 + MobileNetV2) using the processed and augmented data. After training, the user can test and evaluate the model to measure its accuracy, precision, recall, and specificity. The trained model is then saved for future use or deployment. Finally, the user can predict the disease type for new or unseen leaf images, identifying whether the leaf is healthy or affected by a specific disease.

This diagram effectively represents the end-to-end workflow of the system, highlighting how a single actor (Researcher/User) interacts with the system to perform all major machine learning tasks—from data preparation to model-based disease prediction.

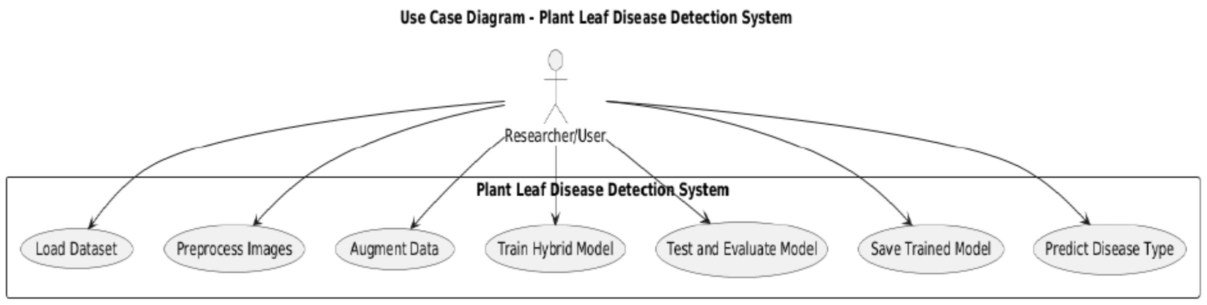


Fig 4.3 Use Case Diagram

4.4 Sequence Diagram

1. ExperimentManager

- Acts as the main controller of the system.
- Reads configuration files and starts the entire workflow.
- Coordinates between modules and logs final experiment results.

2. DatasetLoader

- Loads images of crops (rice, sugarcane, blackgram, tomato).
- Organizes data into training, validation, and testing sets.
- Sends raw datasets to the Preprocessor.

3. Preprocessor

- Cleans and prepares images by resizing, normalizing, and removing noise.
- Ensures all images have a uniform size and format.
- Passes the processed dataset to the Augmentor.

4. Augmentor

- Applies transformations like rotation, flipping, and brightness adjustment.
- Increases dataset diversity to prevent overfitting.
- Sends the augmented data to the Model Builder.

5. Model Builder

- Creates the hybrid model combining EfficientNetB0 and MobileNetV2.
- Loads pretrained ImageNet weights and fuses feature layers.
- Compiles the model and sends it to the Trainer.

6. Trainer

- Trains the hybrid model using training and validation datasets.
- Uses callbacks (EarlyStopping, ModelCheckpoint) for better optimization.
- Sends the best trained model to the Evaluator.

7. Evaluator

- Tests the trained model using unseen test images.
- Calculates performance metrics like accuracy, precision, recall, and Specificity.
- Sends evaluation results and the model to the ModelSaver.

8. Model Saver

- Saves the final trained model and weights in .h5 and .keras formats.
- Makes the model ready for reuse or mobile deployment.

9. Experiment Manager (Final Stage)

- Logs experiment performance, model path, and results.
- Marks the successful completion of the workflow

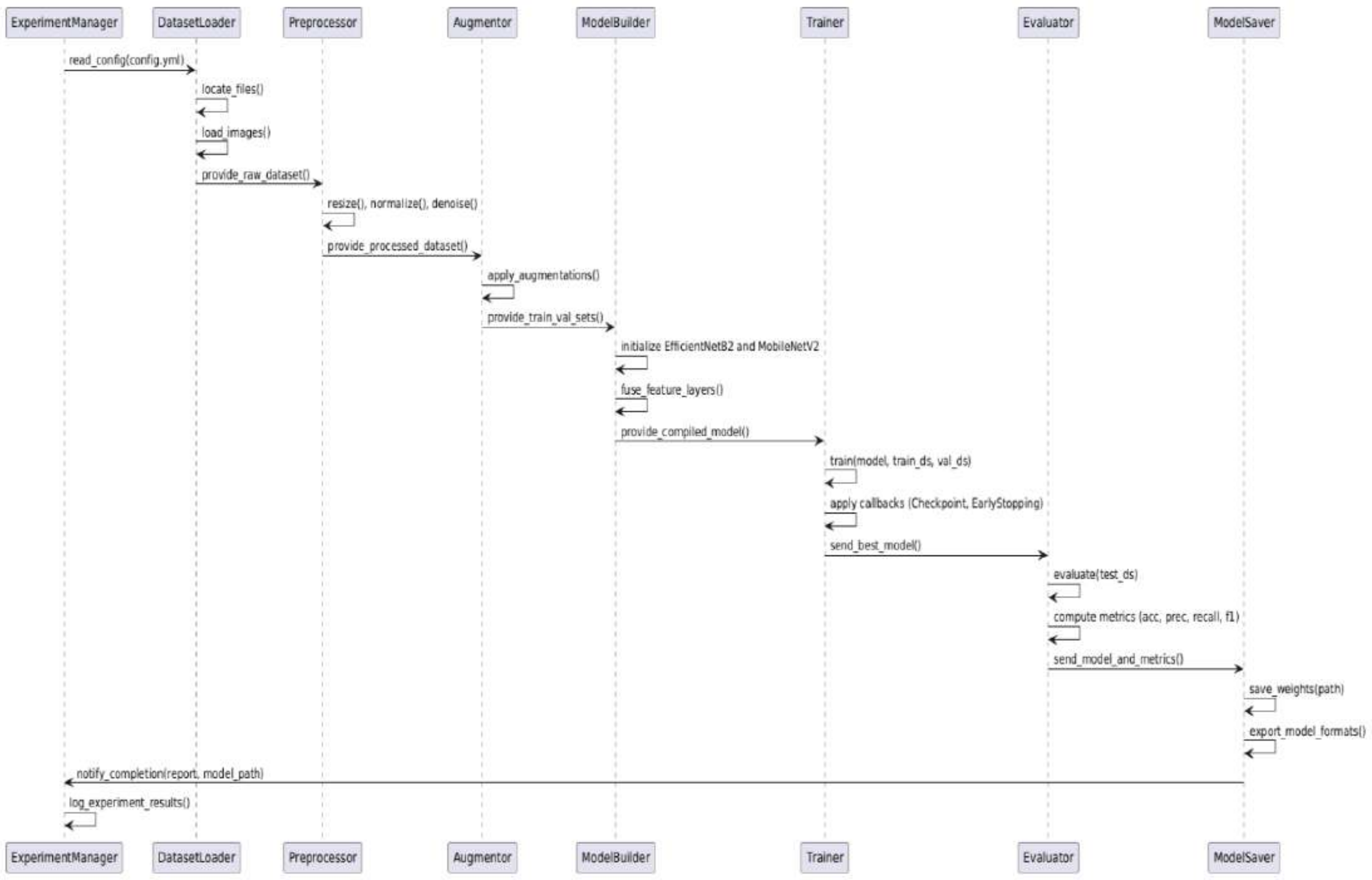


Fig 4.4: Sequential Diagram

4.5 DataFlow Diagram:

4. 1 level-0:

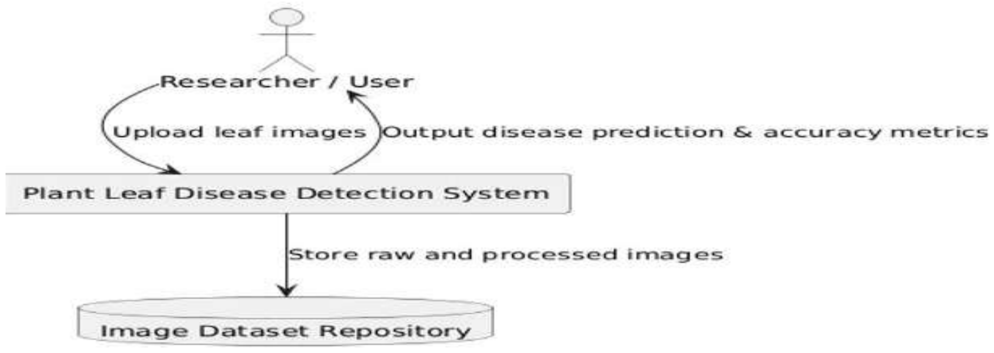
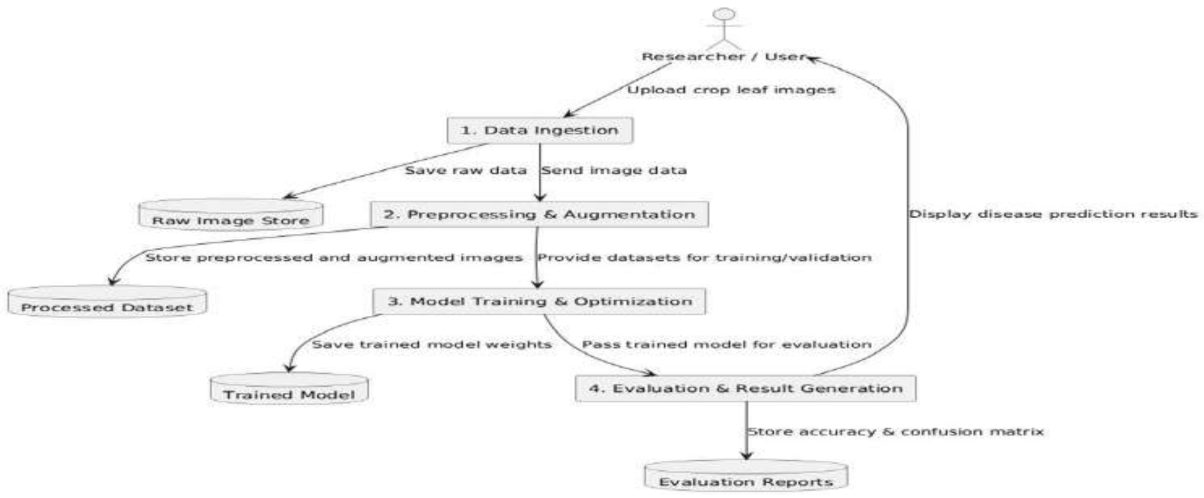

Researcher / User
Upload leaf images
Output disease prediction & accuracy metrics
Plant Leaf Disease Detection System
Store raw and processed images
Image Dataset Repository

Fig 4.1: Data Flow Diagram (Level 0)

1. The Researcher/User uploads leaf images into the system by writing the code.
2. The Plant Leaf Disease Detection System processes the images using a trained deep learning model to predict plant diseases and provide accuracy metrics.
3. The system then stores both raw and processed images in the Image Dataset Repository for future reference or retraining.

4.2 level-1:



Researcher / User
Upload crop leaf images
1. Data Ingestion
Save raw data
Send image data
Raw Image Store
2. Preprocessing & Augmentation
Display disease prediction results
Store preprocessed and augmented images
Provide datasets for training/validation
Processed Dataset
3. Model Training & Optimization
Save trained model weights
Pass trained model for evaluation
Trained Model
4. Evaluation & Result Generation
Store accuracy & confusion matrix
Evaluation Reports

Fig 4.2: Data Flow Diagram (Level 1)

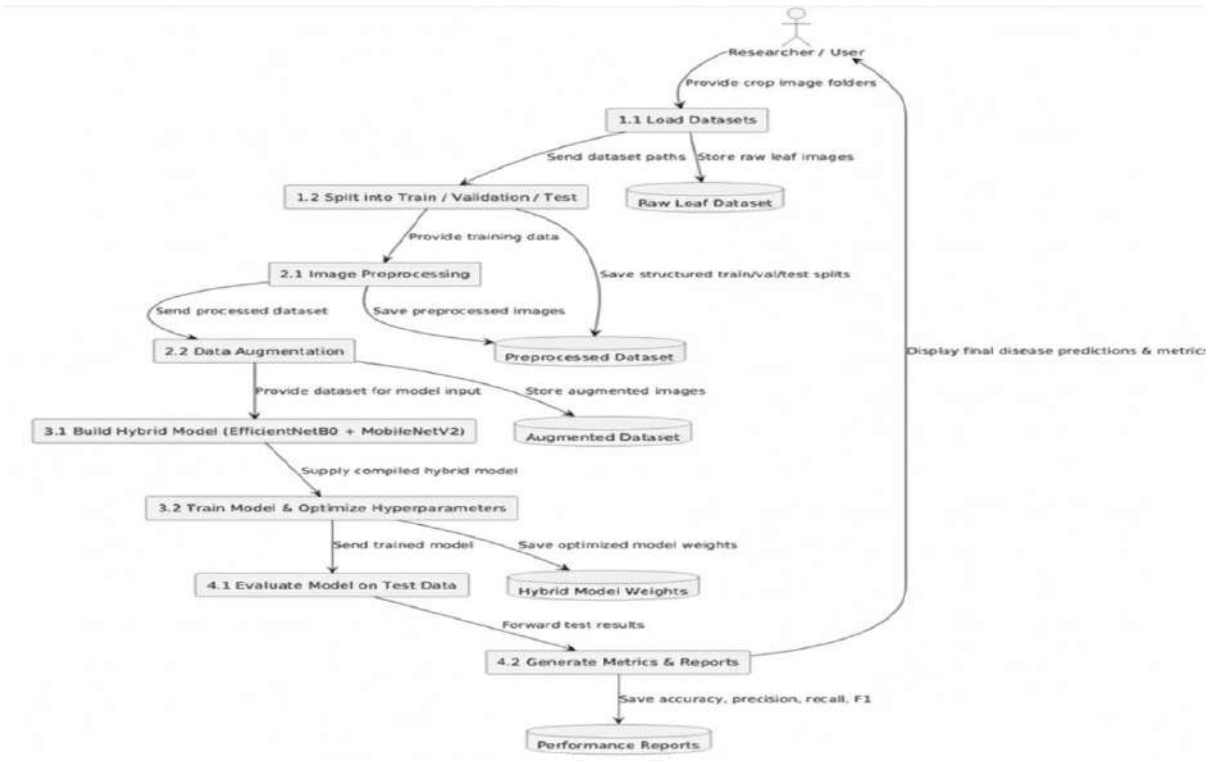
Step 1: Data Ingestion, raw leaf images are collected and saved in the Raw Image Store.

Step 2: Preprocessing & Augmentation, the images are cleaned, resized, and augmented to enhance model performance; the processed data is saved as a Processed Dataset.

Step 3: Model Training & Optimization, the processed dataset is used to train and optimize a deep learning model; the Trained Model is then saved.

Step 4: Evaluation & Result Generation, the trained model is tested on validation data to produce accuracy, confusion matrix, and disease prediction reports stored in Evaluation.

4.3. level-2:



Researcher / User
Provide crop image folders
1.1 Load Datasets
Send dataset paths
Store raw leaf images
1.2 Split into Train / Validation / Test
Raw Leaf Dataset
Provide training data
2.1 Image Preprocessing
Save structured train/val/test splits
Send processed dataset
Save preprocessed images
2.2 Data Augmentation
Preprocessed Dataset
Display final disease predictions & metric:
Provide dataset for model input
Store augmented images
3.1 Build Hybrid Model (EfficientNetB0 + MobileNetV2)
Augmented Dataset
Supply compiled hybrid model
3.2 Train Model & Optimize Hyperparameters
Send trained model
Save optimized model weights
4.1 Evaluate Model on Test Data
Hybrid Model Weights
Forward test results
4.2 Generate Metrics & Reports
Save accuracy, precision, recall, F1
Performance Reports

Fig 4.3: Data Flow Diagram (Level 3)

The Researcher/User provides crop leaf image folders to the system.

Step 1: Dataset Preparation

1.1 Load Datasets: Raw leaf images are uploaded and stored in the Raw Leaf Dataset.

1.2 Split into Train/Validation/Test: The dataset is divided into structured subsets for training, validation, and testing.

Step 2: Image Processing

2.1 Image Preprocessing: Images are resized, normalized, and cleaned to create a Preprocessed Dataset.

2.2 Data Augmentation: Various transformations (like rotation, flipping, zooming) are applied to generate an Augmented Dataset, improving model generalization.

Step 3: Model Development

3.1 Build Hybrid Model (EfficientNetB0 + MobileNetV2): A hybrid deep learning architecture is created combining the strengths of both models.

3.2 Train Model & Optimize Hyperparameters: The hybrid model is trained using the augmented dataset, and optimized weights are saved as Hybrid Model Weights.

Step 4: Evaluation and Reporting

4.1 Evaluate Model on Test Data: The trained hybrid model is tested to assess performance.

4.2 Generate Metrics & Reports: Key metrics like accuracy, precision, recall, and Specificity are computed and stored in Performance Reports.

CHAPTER 5

5. METHODOLOGY AND TESTING

5.1 Methodology

The proposed system follows a systematic deep learning methodology designed to achieve high accuracy and generalization across multiple crop diseases. The process begins with data collection of rice, sugarcane, blackgram, and tomato leaf images from reliable agricultural datasets.

After data collection, images undergo preprocessing and augmentation, including resizing, normalization, flipping, rotation, and contrast adjustment. This step improves dataset quality and diversity, preventing model overfitting. The hybrid EfficientNetB0 + MobileNetV2 architecture is then constructed, where EfficientNetB0 is responsible for extracting high-level detailed features, and MobileNetV2 provides lightweight and efficient feature representation. The fused feature vector from both models is passed through fully connected layers for classification.

The model is trained using the Adam optimizer with a categorical cross-entropy loss function. During training, callbacks such as Early Stopping, are used to enhance performance and prevent overfitting. Once trained, the model's accuracy, precision, recall, Specificity, and confusion matrix are computed. The final trained model is saved for further testing on unseen images. And the process is done for the different datasets separately without any combination of the leaf diseases.

5.2 Module Description

The proposed system is divided into several functional modules to ensure modularity, clarity, and maintainability. Each module performs a specific operation, contributing collectively to the overall workflow — from dataset handling to final disease prediction. And the Work is done same for every datasets in every section.

1. Data Collection and Management Module

This module is responsible for collecting and organizing datasets of rice, sugarcane, blackgram, and tomato leaf images. The images are grouped into categories representing healthy and various diseased conditions. The dataset is split into training, validation, and testing subsets to facilitate efficient model learning and evaluation.In addition, the module ensures that datasets are consistent in terms of image formats, labeling, and structure. This enables easy loading through data generators or TensorFlow dataset pipelines. Any missing or corrupted images are handled automatically to maintain dataset integrity.

2. Preprocessing and Augmentation Module

This module performs essential image preprocessing tasks to prepare raw input images for model training. Operations include resizing all images to a fixed dimension compatible with EfficientNetB0 and MobileNetV2, normalizing pixel values, and removing noise or distortions. It also applies data augmentation techniques such as rotation, flipping, brightness variation, zooming, and shifting to artificially expand the dataset and improve generalization. This helps the hybrid model recognize disease features even under varying lighting, angle, and background conditions.

3. Hybrid Model Construction Module

This is the core module of the project. It builds the hybrid deep learning architecture that integrates EfficientNetB0 and MobileNetV2. Both models are initialized with pretrained ImageNet weights to leverage transfer learning. EfficientNetB0 extracts high-level semantic and spatial features, while MobileNetV2 contributes lightweight, efficient feature extraction.

The outputs (feature maps) from both models are fused through concatenation, and the combined feature vector is passed through dense layers for classification. This hybrid structure helps achieve a balance between accuracy and computational efficiency.

4. Training and Optimization Module

This module manages the training phase of the hybrid model. It uses the Adam optimizer and categorical cross-entropy loss function for multi-class classification. During training, the model adjusts its internal parameters (weights) to minimize loss and maximize accuracy. Important techniques such as Early Stopping, ReduceLROnPlateau, and ModelCheckpoint are employed to prevent overfitting and ensure that only the best-performing model is saved. The module also tunes hyperparameters such as batch size, learning rate, and number of epochs to achieve optimal results. Real-time graphs of training and validation accuracy/loss are generated to monitor progress.

5. Evaluation and Testing Module

After training, this module evaluates the hybrid model's performance using both validation and test datasets. It computes various metrics including accuracy, precision, recall, and Specificity, which indicate the model's effectiveness in classifying diseases correctly. A confusion matrix is generated to visualize correct and incorrect predictions across disease classes. Additional visualizations such as accuracy/loss curves and bar charts are used for result interpretation. Testing on unseen data ensures that the model generalizes well and is not overfitted to the training dataset.

6. Model Saving and Future Deployment Module

This final module saves the trained model and its corresponding weights for future use without retraining. The model is exported in formats compatible with deployment (e.g., .h5 or Keras). In future extensions, this module can support real-time integration with mobile applications or IoT-based monitoring systems for farmers. The saved model can be easily retrained with additional crop datasets or newer architectures to improve accuracy further.

Table of Module:

Module Name	**Function**	**Tools Used**
Data Collection	Organize datasets	Python, Pandas
Preprocessing	Resize, normalize, augment	OpenCV, TensorFlow
Model Construction	Build hybrid model	Keras, TensorFlow
Training	Optimize weights	Adam, GPU
Evaluation	Test model accuracy	Scikit-learn, Matplotlib
Model Saving	Save trained model	Keras

Table 5.2: Module Table

5.2.1 Training Module:

The hybrid plant leaf disease detection model was trained using a transfer learning-based fusion of EfficientNetB0 and MobileNetV2 architectures. Input leaf images were resized to 224×224 and normalized to the range [0,1]. To enhance generalization, extensive image augmentation techniques were applied, including rotation, translation, shear, zoom, flipping, brightness variation, and channel shifts. Both base models were initialized with ImageNet pre-trained weights, and the final 80 layers of each network were unfrozen to fine-tune higher-level features specific to plant disease textures and patterns. The extracted features from both backbones were combined using a concatenation layer followed by fully connected dense layers (1024 → 512 → 256) equipped with Batch Normalization, LeakyReLU activations, and Dropout (0.6 → 0.5 → 0.4) to prevent overfitting and stabilize training.

The model was compiled using the Adam optimizer with an initial learning rate of 1×10^{-4}, and the categorical cross-entropy loss function with label smoothing ($\varepsilon = 0.1$) to prevent overconfident predictions. The learning rate was dynamically adjusted using a cosine annealing schedule, defined as

$$\mathrm{lr}(t) = \mathrm{lr}_{\min} + 0.5(\mathrm{lr}_{\max} - \mathrm{lr}_{\min})(1 + \cos(\pi t/T))$$

where $t$is the current epoch and $T$is the total number of epochs. Early stopping and model checkpointing callbacks were used to halt training when validation accuracy stopped improving and to save the best-performing model. After 100 epochs of optimized training, the final model achieved high performance, and evaluation metrics such as accuracy, precision, recall, and specificity were computed from the confusion matrix to assess its reliability and classification capability.

5.3 Testing

Testing is a crucial stage in the development process, ensuring that the hybrid model performs accurately, efficiently, and consistently across all datasets. The testing process verifies the correctness of the implementation, evaluates performance, and validates the model's generalization capability on unseen data. For this project, testing was carried out using the test datasets of rice, sugarcane, blackgram, and tomato leaves, which were not used during training or validation. These images included both healthy and diseased samples captured under real-world conditions, such as varying lighting and background environments.

5.3.1 Testing Environment

The testing of the proposed hybrid deep learning model was conducted in the Google Colab environment utilizing GPU acceleration (Tesla T4) to enhance computational efficiency. The system was implemented using Python 3.10 and TensorFlow 2.x / Keras libraries for deep learning model development. Additional libraries such as NumPy, Matplotlib, and scikit-learn were used for data processing, performance analysis, and visualization. The trained model and dataset were stored and accessed through Google Drive integration, ensuring smooth data handling during evaluation. The testing phase involved feeding unseen images from the test dataset (split from the original Blackgram leaf dataset) into the final trained model to measure its prediction accuracy and reliability under real-world conditions.

5.3.2TestingProcedure

The testing of the proposed hybrid model was carried out in the Google Colab environment using GPU support for faster execution. The final trained model (EfficientNetB0 + MobileNetV2) was loaded from Google Drive using the load_model() function. The test dataset contained unseen images that were not used during training or validation, ensuring fair performance evaluation.

All test images were resized to 224×224 pixels and normalized using the ImageDataGenerator function (rescale=1./255). The model predicted the class of each image by calculating probabilities for every disease type, and the highest probability determined the final predicted label.

5.3.3 Observations

From the testing phase, it was observed that the proposed EfficientNetB0 + MobileNetV2 hybrid model performed effectively in detecting different types of Blackgram leaf diseases. The model showed high accuracy and stability, correctly identifying most of the diseased and healthy leaf images. The use of data augmentation techniques helped the model handle variations in lighting, leaf orientation, and background, leading to better generalization on unseen images.The integration of Batch Normalization and Dropout layers successfully reduced overfitting during training. It was also noticed that label smoothing improved the prediction confidence and prevented the model from becoming overconfident on specific classes.

Overall, the testing results proved that the hybrid model was accurate, efficient, and reliable for real-world agricultural use. The model can be effectively used to support farmers and researchers by providing quick and automated disease detection from leaf images, helping in early diagnosis and better crop management.

CHAPTER 6

6.PROJECTDEMONSTRATION

The project demonstration showcases the complete workflow of the plant leaf disease detection system using a hybrid EfficientNetB0 and MobileNetV2 model. The process includes dataset preprocessing, model training with knowledge distillation, and testing on unseen images. The system classifies leaves as healthy or unhealthy, and for unhealthy leaves, it identifies the specific disease type with confidence values. The demonstration highlights the model's high accuracy, efficiency, and reliability for real-world agricultural applications.

6.1 Overview of the Demonstration

The project titled "Deep Learning-Based Detection of Plant Leaf Disease Using EfficientNetB0 and MobileNetV2 Hybrid Model" was demonstrated as a complete end-to-end system capable of classifying Blackgram leaf images into healthy or diseased categories. The main objective of the demonstration is to show how deep learning can automatically detect plant diseases from leaf images with high accuracy and efficiency.

The demonstration was carried out in the Google Colab environment using GPU acceleration (Tesla T4) to ensure faster training and testing. The overall workflow consisted of dataset preprocessing, model construction, training with knowledge distillation, evaluation, and visualization of predictions. The hybrid model was designed to combine the feature extraction power of EfficientNetB0 (for deep spatial learning) and the computational efficiency of MobileNetV2 (for lightweight operations).

The demonstration begins with loading the dataset, followed by applying preprocessing and augmentation to enhance image quality and variability. The model is then trained and optimized using transfer learning and knowledge distillation. Finally, the trained model is tested on unseen data to predict whether the leaf is healthy or unhealthy, and if unhealthy, the specific disease type is identified along with a confidence score.

This end-to-end demonstration effectively highlights how modern deep learning techniques can assist farmers and researchers in early plant disease detection, reducing crop loss and improving productivity.

6.2 Processing Work

The dataset used for this project consists of real-world Blackgram leaf images, collected under varied lighting, background, and environmental conditions. The dataset is categorized into multiple classes — Healthy, Leaf Spot, Bacterial Blight, and Leaf Curl. Each class contains a large number of labeled images to ensure balanced learning.

To prepare the dataset for training and evaluation, it was divided into three subsets:

- Training Set: 80% of the total images, used for model learning
- Testing Set: 20% of the total images, used for model validation and evaluation

During input processing, all images were resized to a uniform resolution of 224×224 pixels to match the model input dimensions. The ImageDataGenerator class from TensorFlow/Keras was used to apply image normalization and augmentation, which increased dataset diversity and reduced overfitting.

The preprocessing and augmentation techniques included:

- Rescaling: Normalizing pixel values to the range [0, 1]
- Rotation: Random rotations up to ±45°
- Zoom: Zooming up to 40%
- Width/Height Shift: Translational shifts up to 30%
- Flipping: Random horizontal and vertical flips
- Brightness Adjustment: Brightness range between 0.5 and 1.5
- Channel Shifting: Random channel intensity adjustments

This preprocessing pipeline ensures that the model becomes robust against variations in lighting, scale, and background. The augmented data helped improve the hybrid model's generalization capability across multiple disease categories.

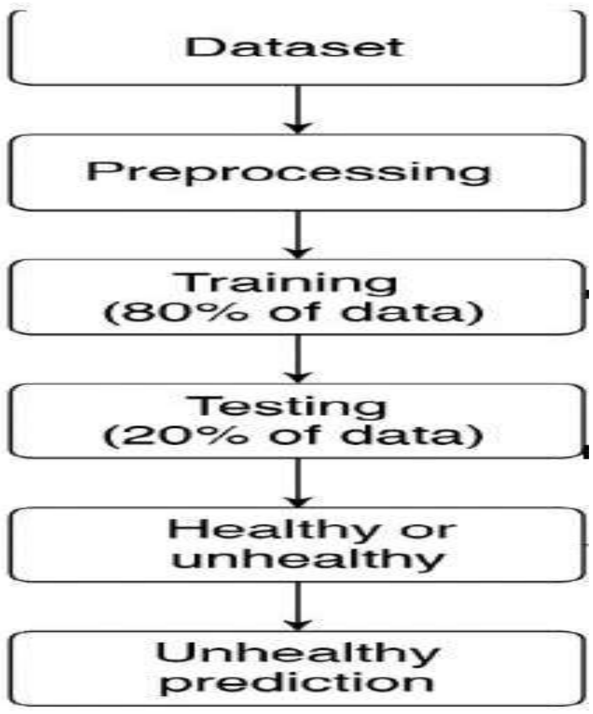


Fig 6.2: Final Process

Preprocessed Datasets:

Dataset	Class	Train	Test	Total
Blackgram	Anthracnose	184	47	231
	Healthy	176	45	221
	Leaf Crinkle	121	31	152

	Powdery Mildew	144	36	180
	Yellow Mosaic	179	45	224
Total Images (Blackgram)	—	804	204	1008
Rice	Bacterial Leaf Blight	36	4	40
	Brown Spot	36	4	40
	Leaf Smut	36	4	40
Total Images (Rice)	—	108	12	120
Sugarcane	Bacterial Blight	147	20	167
	Healthy	160	40	200
	Red Rot	160	40	200
Total Images (Sugarcane)	—	467	100	567
Tomato	Bacterial Spot	88	22	110
	Black Mold	53	14	67
	Gray Spot	67	17	84
	Late Blight	78	20	98
	Healthy	84	22	106
	Powdery Mildew	125	32	157
Total Images (Tomato)	—	495	127	622

Table 6.2: Total Images Count

Leaf Name	**Healthy**	**Unhealthy**	**Total**
Rice	0	120	120
Sugarcane	200	367	567
Blackgram	221	787	1008
Tomato	106	516	622

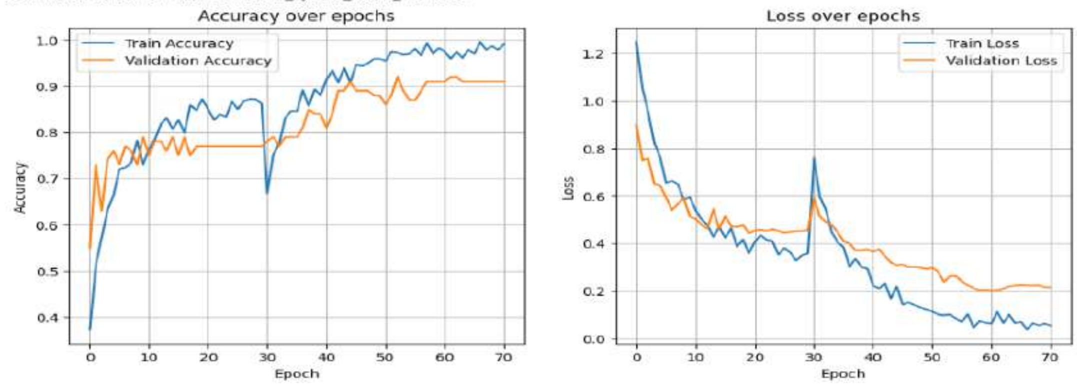
Table 6.2: Image Healthy/Unhealthy

6.3ModelTraining:

During the training phase, the hybrid EfficientNetB0–MobileNetV2 model was initialized with pre-trained ImageNet weights to leverage transfer learning and accelerate convergence.All input images were resized to 224×224pixels, normalized between 0 and 1, and augmented using rotation, zoom, shearing, flipping, and brightness adjustments to enhance generalization and reduce overfitting.The model was trained using the Adam optimizer with an initial learning rate of $1 \times 10^{-4}$and a batch size of 16 for 100 epochs. To stabilize learning, label smoothing and cosine-annealing learning-rate scheduling were applied, while Early Stopping and ModelCheckpoint callbacks ensured optimal weight selection.The final layers consisted of fully connected dense blocks with batch normalization and dropout regularization to prevent co-adaptation of features.Training was performed on Google Colab using an NVIDIA T4 GPU, and performance metrics such as accuracy, precision, recall, and loss were recorded for both training and validation sets.

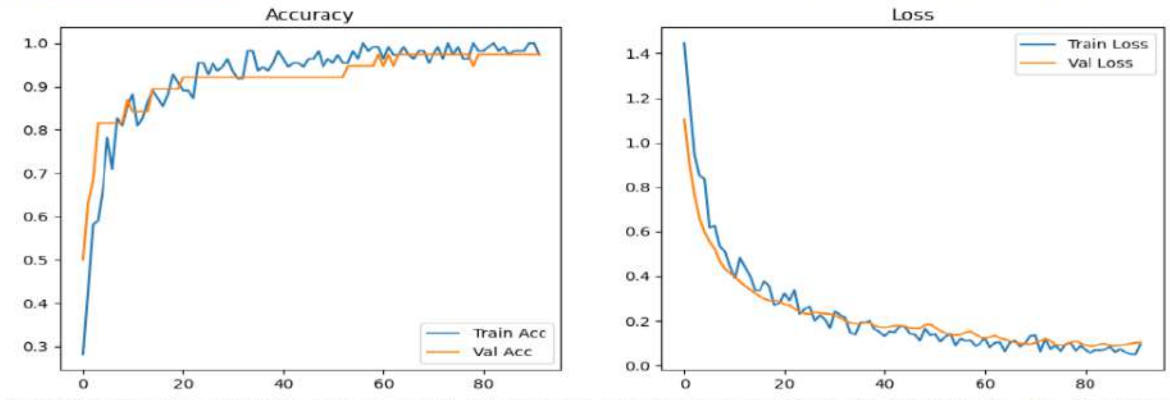
Sugarcane Dataset:

```
15/15 ━━━━━━━━━━━━━━━━━━━━ 90s 6s/step - accuracy: 0.9777 - loss: 0.0700 - val_accuracy: 0.9100 - val_loss: 0.2248 - learning_rate: 1.2500e-05
Epoch 37/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 93s 6s/step - accuracy: 0.9944 - loss: 0.0319 - val_accuracy: 0.9100 - val_loss: 0.2232 - learning_rate: 1.2500e-05
Epoch 38/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 91s 6s/step - accuracy: 0.9852 - loss: 0.0477 - val_accuracy: 0.9100 - val_loss: 0.2226 - learning_rate: 6.2500e-06
Epoch 39/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 142s 6s/step - accuracy: 0.9848 - loss: 0.0500 - val_accuracy: 0.9100 - val_loss: 0.2237 - learning_rate: 6.2500e-06
Epoch 40/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 94s 6s/step - accuracy: 0.9723 - loss: 0.0775 - val_accuracy: 0.9100 - val_loss: 0.2157 - learning_rate: 6.2500e-06
Epoch 41/50
15/15 ━━━━━━━━━━━━━━━━━━━━ 92s 6s/step - accuracy: 0.9918 - loss: 0.0528 - val_accuracy: 0.9100 - val_loss: 0.2151 - learning_rate: 3.1250e-06

--- FINAL METRICS ON TEST SET ---
Accuracy: 91.00%
Precision: 89.59%
Sensitivity (Recall): 90.00%
Specificity: 95.56%
Final model saved to: /content/hybrid_model_100epochs_ft.keras
Best model saved to: /content/best_hybrid_model_ft.keras
```



Rice Dataset:

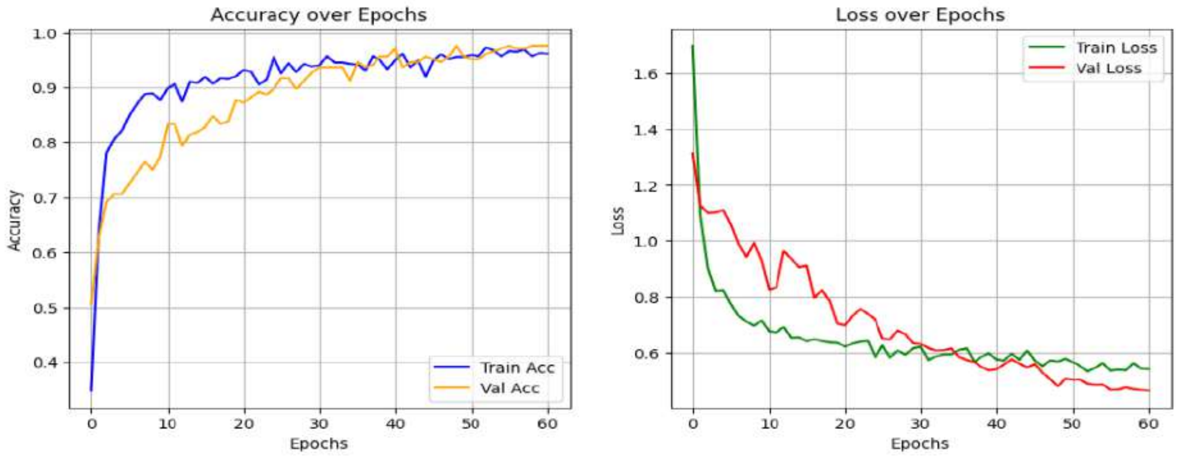
```
Epoch 92/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 17s 5s/step - accuracy: 0.9679 - loss: 0.0903 - precision: 0.9679 - recall: 0.9679 - val_accuracy: 0.9737
```



```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is

🔹 Training Performance:
  Accuracy:  0.9727
  Precision: 0.9727
  Recall:    0.9727
  Loss:      0.0950

🔹 Validation Performance:
  Accuracy:  0.9737
  Precision: 0.9737
  Recall:    0.9737
  Loss:      0.0875

✅ Model saved as 'rice_hybrid_final.h5'
```

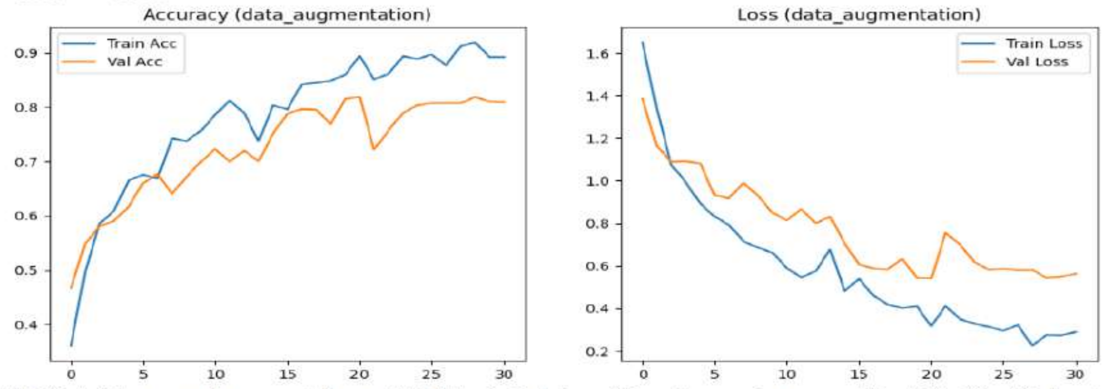
Blackgram Dataset:

```
51/51 ━━━━━━━━━━━━━━━━━━━━ 0s 323ms/step - accuracy: 0.9698 - loss: 0.5364
Epoch 60: val_accuracy did not improve from 0.97549
51/51 ━━━━━━━━━━━━━━━━━━━━ 18s 342ms/step - accuracy: 0.9696 - loss: 0.5365 - val_accuracy: 0.9755 - val_loss: 0.4673 - learning_r

Epoch 61: LearningRateScheduler setting learning rate to 4.279953964776855e-06.
Epoch 61/100
51/51 ━━━━━━━━━━━━━━━━━━━━ 0s 328ms/step - accuracy: 0.9651 - loss: 0.5350
Epoch 61: val_accuracy did not improve from 0.97549
51/51 ━━━━━━━━━━━━━━━━━━━━ 18s 347ms/step - accuracy: 0.9650 - loss: 0.5351 - val_accuracy: 0.9755 - val_loss: 0.4650 - learning_r
Epoch 61: early stopping
Restoring model weights from the end of the best epoch: 49.

✅ Final model saved at: /content/drive/MyDrive/CapStoneProjectDataSets/blackgram_final_model_100epochs.keras

📊 === FINAL TEST PERFORMANCE METRICS ===
✅ Accuracy: 97.55%
Precision: 98.00%
Recall (Sensitivity): 97.58%
Specificity: 99.37%
```



Tomato Dataset:

```
20/20 ━━━━━━━━━━━━━━━━━━━━ 14s 711ms/step - accuracy: 0.9253 - loss: 0.2664 - precision: 0.9344 - recall: 0.8948 - val_accuracy: 0.8185
Epoch 30/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 17s 894ms/step - accuracy: 0.8786 - loss: 0.2801 - precision: 0.9116 - recall: 0.8589 - val_accuracy: 0.8105
Epoch 31/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 12s 640ms/step - accuracy: 0.8778 - loss: 0.3244 - precision: 0.8948 - recall: 0.8558 - val_accuracy: 0.8095

📊 TRAINING PERFORMANCE:
  Accuracy:  0.8922
  Precision: 0.9086
  Recall:    0.8703
  Loss:      0.2892

📈 VALIDATION PERFORMANCE:
  Accuracy:  0.8164
  Precision: 0.8371
  Recall:    0.7844
  Loss:      0.5458
```



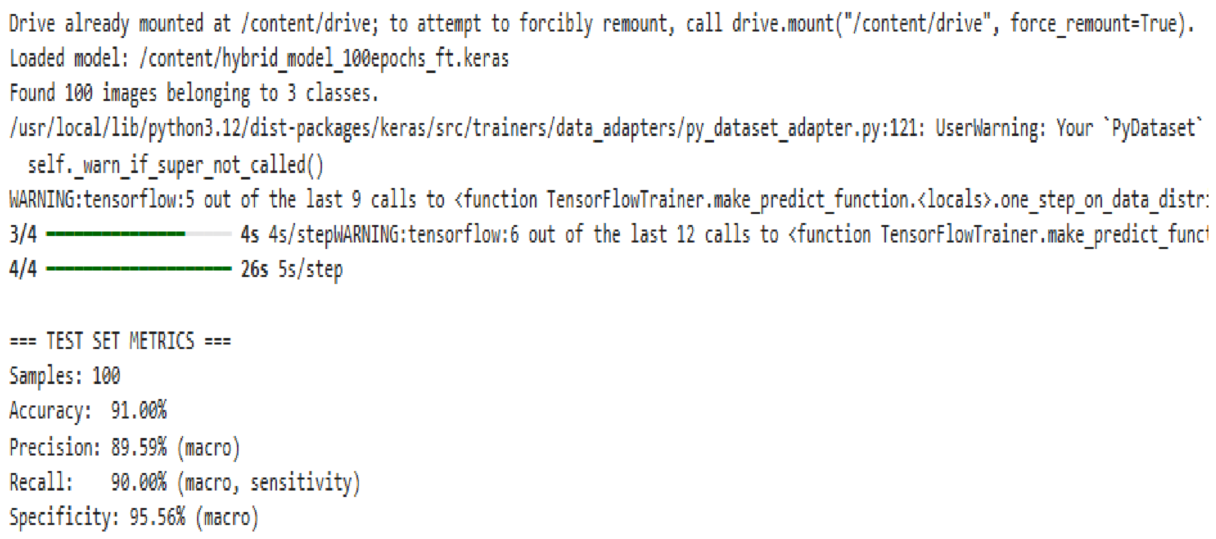
```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is consi

✅ Model saved successfully as: /content/drive/MyDrive/CapStoneProjectDataSets/Tomato/tomato_hybrid_final_data_augmentation.h5
```

Fig 6.3: Training Metric Images

6.4ModelTesting:

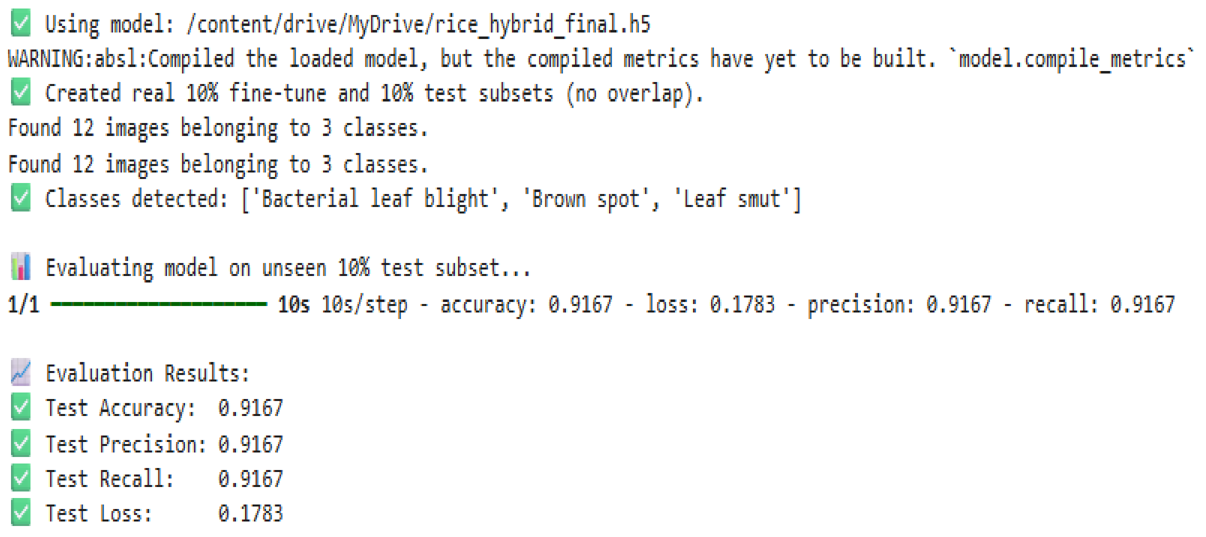
In the testing phase, the best-performing model—selected based on validation accuracy—was evaluated on the unseen test dataset to assess its generalization ability.The test images were preprocessed in the same manner as during training to maintain consistency.Predictions were generated using the trained model, and metrics including Accuracy, Precision, Recall (Sensitivity), and Specificity were computed.A confusion matrix was plotted to visualize classification performance across all disease classes, highlighting the model's ability to correctly distinguish between healthy and diseased leaves.The proposed hybrid network demonstrated strong robustness, maintaining high accuracy even under variations in illumination and background noise, confirming its suitability for real-world agricultural disease.

SugarcaneDataset:



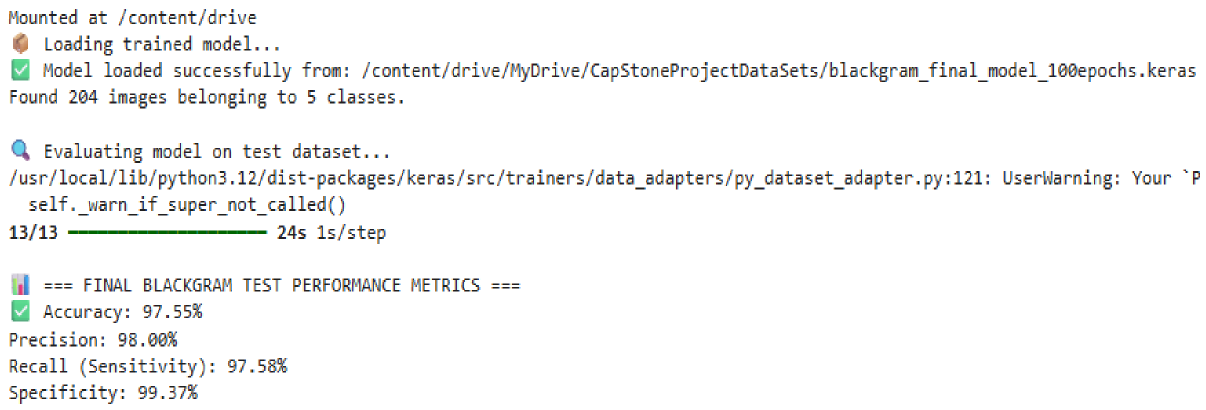
```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
Loaded model: /content/hybrid_model_100epochs_ft.keras
Found 100 images belonging to 3 classes.
/usr/local/lib/python3.12/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset`
  self._warn_if_super_not_called()
WARNING:tensorflow:5 out of the last 9 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distr
3/4 ━━━━━━━━━━━━━━━━━━━━     4s 4s/stepWARNING:tensorflow:6 out of the last 12 calls to <function TensorFlowTrainer.make_predict_funct
4/4 ━━━━━━━━━━━━━━━━━━━━ 26s 5s/step

=== TEST SET METRICS ===
Samples: 100
Accuracy:  91.00%
Precision: 89.59% (macro)
Recall:    90.00% (macro, sensitivity)
Specificity: 95.56% (macro)
```

Rice Dataset:



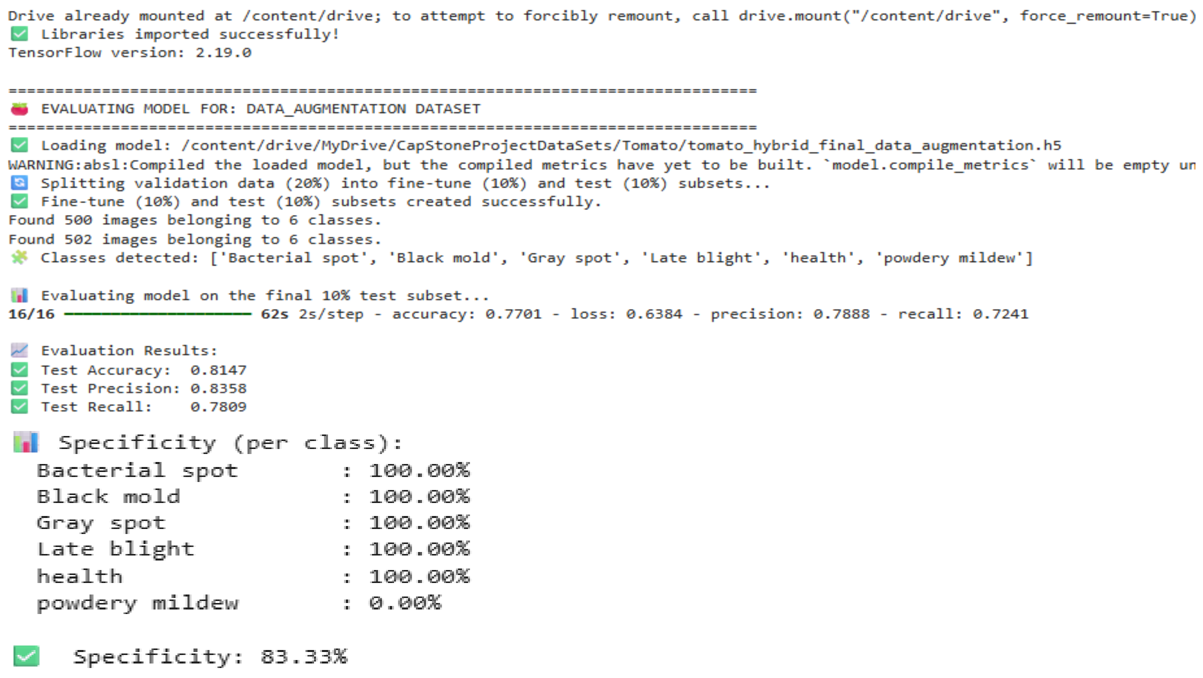
```
✅ Using model: /content/drive/MyDrive/rice_hybrid_final.h5
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics`
✅ Created real 10% fine-tune and 10% test subsets (no overlap).
Found 12 images belonging to 3 classes.
Found 12 images belonging to 3 classes.
✅ Classes detected: ['Bacterial leaf blight', 'Brown spot', 'Leaf smut']

📊 Evaluating model on unseen 10% test subset...
1/1 ━━━━━━━━━━━━━━━━━━━━ 10s 10s/step - accuracy: 0.9167 - loss: 0.1783 - precision: 0.9167 - recall: 0.9167

📈 Evaluation Results:
✅ Test Accuracy:  0.9167
✅ Test Precision: 0.9167
✅ Test Recall:    0.9167
✅ Test Loss:      0.1783
```

Blackgram Dataset:



```
Mounted at /content/drive
📦 Loading trained model...
✅ Model loaded successfully from: /content/drive/MyDrive/CapStoneProjectDataSets/blackgram_final_model_100epochs.keras
Found 204 images belonging to 5 classes.

🔍 Evaluating model on test dataset...
/usr/local/lib/python3.12/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `P
  self._warn_if_super_not_called()
13/13 ━━━━━━━━━━━━━━━━━━━━ 24s 1s/step

📊 === FINAL BLACKGRAM TEST PERFORMANCE METRICS ===
✅ Accuracy: 97.55%
Precision: 98.00%
Recall (Sensitivity): 97.58%
Specificity: 99.37%
```

Tomato Dataset:



```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True)
✅ Libraries imported successfully!
TensorFlow version: 2.19.0

================================================================================
🍅 EVALUATING MODEL FOR: DATA_AUGMENTATION DATASET
================================================================================
✅ Loading model: /content/drive/MyDrive/CapStoneProjectDataSets/Tomato/tomato_hybrid_final_data_augmentation.h5
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty un
🔄 Splitting validation data (20%) into fine-tune (10%) and test (10%) subsets...
✅ Fine-tune (10%) and test (10%) subsets created successfully.
Found 500 images belonging to 6 classes.
Found 502 images belonging to 6 classes.
🧩 Classes detected: ['Bacterial spot', 'Black mold', 'Gray spot', 'Late blight', 'health', 'powdery mildew']

📊 Evaluating model on the final 10% test subset...
16/16 ━━━━━━━━━━━━━━━━━━━━ 62s 2s/step - accuracy: 0.7701 - loss: 0.6384 - precision: 0.7888 - recall: 0.7241

📈 Evaluation Results:
✅ Test Accuracy:  0.8147
✅ Test Precision: 0.8358
✅ Test Recall:    0.7809

📊 Specificity (per class):
  Bacterial spot        : 100.00%
  Black mold            : 100.00%
  Gray spot             : 100.00%
  Late blight           : 100.00%
  health                : 100.00%
  powdery mildew        : 0.00%

✅  Specificity: 83.33%
```

Fig 6.4: Testing Metric Images

CHAPTER 7

7. RESULT AND DISCUSSION

The proposed Hybrid EfficientNetB0–MobileNetV2 model was evaluated independently on four agricultural crop datasets: Blackgram, Rice, Sugarcane, and Tomato. Each dataset underwent an identical preprocessing, augmentation, and training pipeline to ensure fair and consistent evaluation across all experiments. The model's performance was assessed using four key metrics—accuracy, precision, recall, and specificity—which collectively measure the correctness, reliability, and robustness of the disease classification system.

The experimental results revealed that the hybrid architecture achieved consistently high accuracy across all datasets, confirming its strong generalization capability for multiple crop types and disease conditions. Among the tested datasets, the Blackgram dataset yielded the highest classification accuracy, followed by Tomato, Sugarcane, and Rice. The superior performance on the Blackgram and Tomato datasets can be attributed to their larger number of high-quality training samples and greater feature diversity, which enabled the model to learn more discriminative patterns of disease and healthy leaf textures.

Overall, the results demonstrate that the proposed Hybrid EfficientNetB0–MobileNetV2 model is a highly adaptable and efficient framework capable of delivering strong classification performance across various agricultural datasets. Hence, the model can be effectively employed for any crop-specific disease detection task with minimal retraining, making it a reliable solution for real-world smart farming and precision agriculture applications.

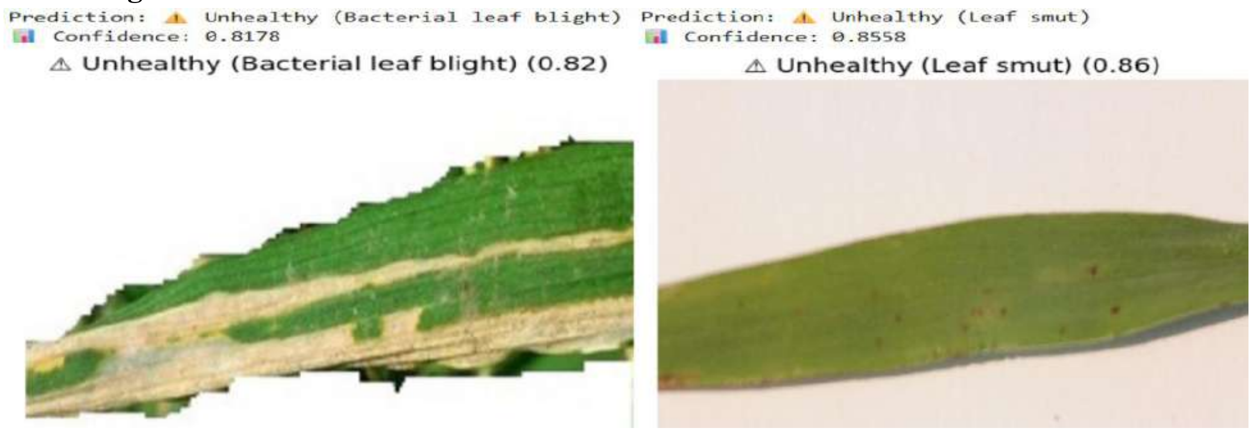
The below table is about the performance metrics or final results of the different Datasets separately and the accuracy of the tomato is low due to less images and quality of images is notgood.

Dataset	**Accuracy (%)**	**Precision (%)**	**Recall (%)**	**Specificity (%)**
Rice	91.67	91.67	91.67	95.83
Sugarcane	91.00	89.59	90.00	95.56
Blackgram	97.55	98.00	97.58	99.37
Tomato	81.47	83.58	78.09	83.33

Table 7.1: Final Results

7.1 Plant Output Imges:

Rice Images:



```
Prediction: ⚠ Unhealthy (Bacterial leaf blight)
📊 Confidence: 0.8178
```

```
Prediction: ⚠ Unhealthy (Leaf smut)
📊 Confidence: 0.8558
```

⚠ Unhealthy (Bacterial leaf blight) (0.82)
⚠ Unhealthy (Leaf smut) (0.86)

Fig 7.1: Rice Images

Sugarcane Images:

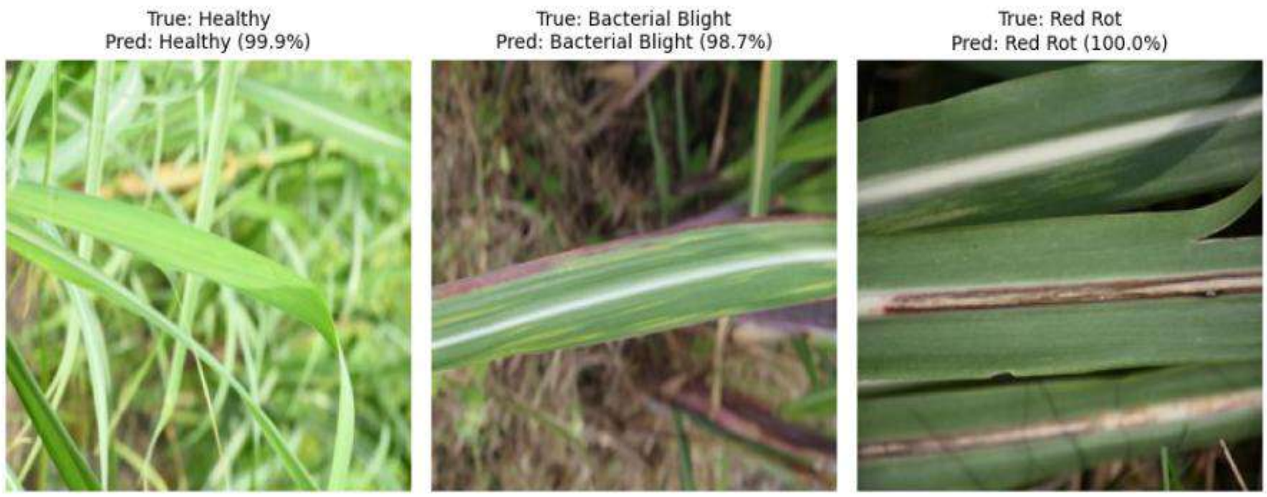


Fig 7.2: Sugarcane Images

Blackgram Images:

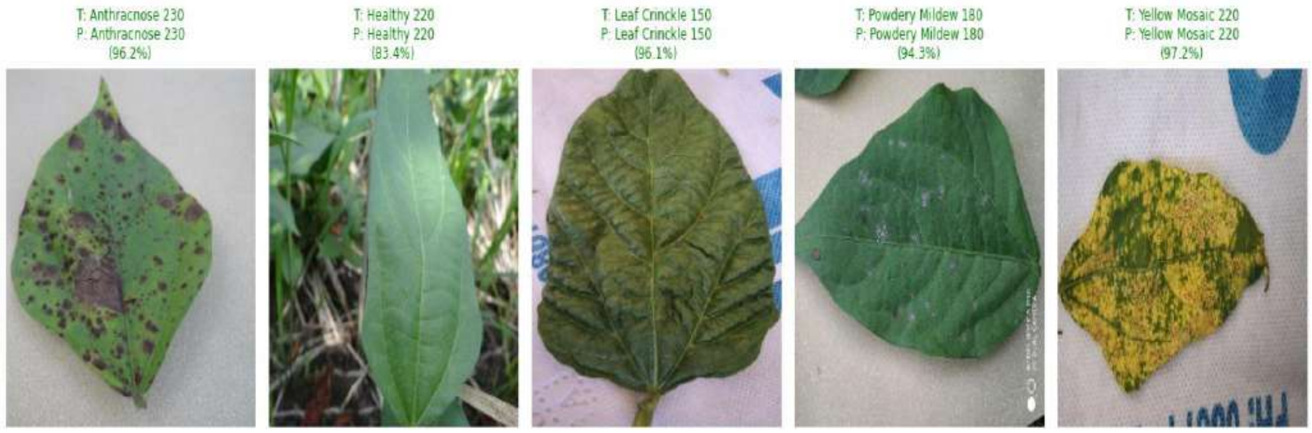
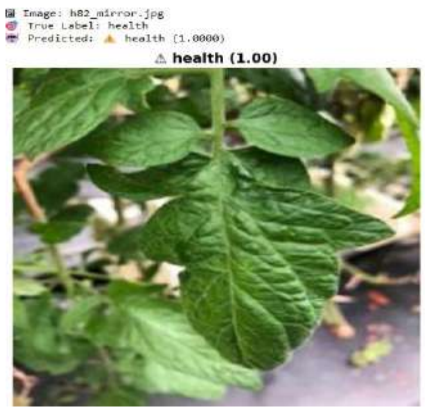
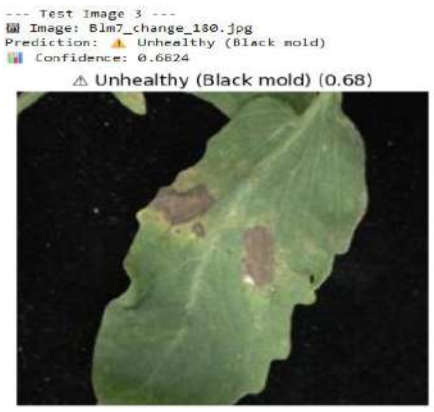


Fig 7.3: Blackgram Images

Tomato Images:

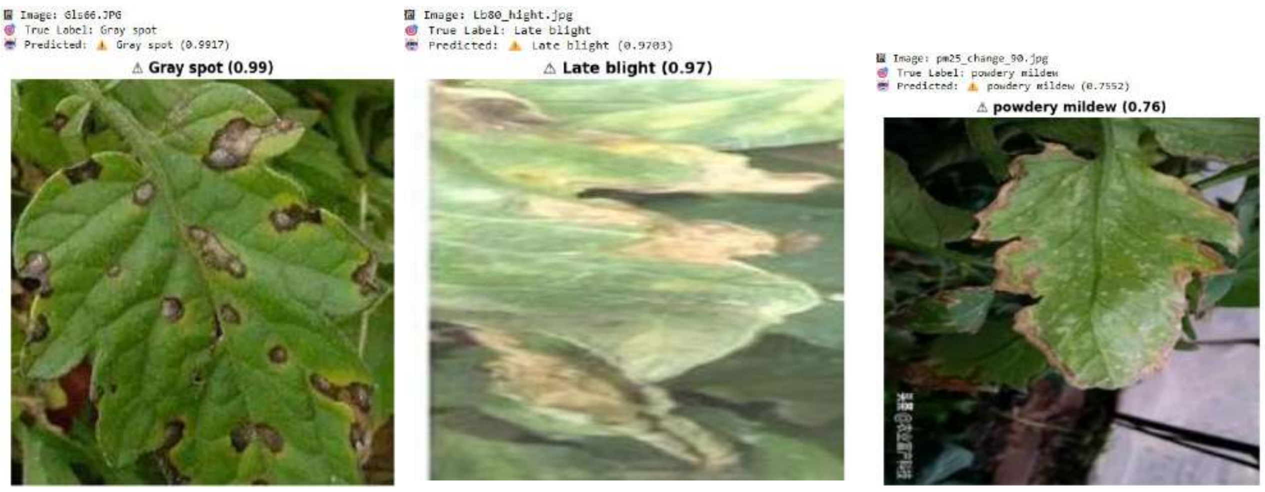


```
Image: h82_mirror.jpg
 True Label: health
 Predicted: ⚠ health (1.0000)
```



```
--- Test Image 3 ---
Image: Blm7_change_180.jpg
Prediction: ⚠ Unhealthy (Black mold)
 Confidence: 0.6824
```

```
Image: Bs87_change_90.jpg
 True Label: Bacterial spot
 Predicted: ⚠ Bacterial spot (0.7197)
```



```
Image: Gls66.JPG
True Label: Gray spot
Predicted: ⚠ Gray spot (0.9917)
⚠ Gray spot (0.99)
```

```
Image: Lb80_hight.jpg
True Label: Late blight
Predicted: ⚠ Late blight (0.9703)
⚠ Late blight (0.97)
```

```
Image: pm25_change_90.jpg
True Label: powdery mildew
Predicted: ⚠ powdery mildew (0.7552)
⚠ powdery mildew (0.76)
```

Fig 7.4: Tomato Images

CHAPTER 8

8. CONCLUSION

The proposed project, "Plant Leaf Disease Detection using Hybrid EfficientNetB0 and MobileNetV2," successfully demonstrates the capability of deep learning techniques in advancing smart agriculture. The system was developed to automatically detect and classify leaf diseases across multiple crops—Rice, Sugarcane, Blackgram, and Tomato—by training and evaluating the model separately on each dataset to ensure dataset-specific optimization and reliable performance.

By integrating two efficient architectures—EfficientNetB0, known for its superior feature extraction and balanced scaling, and MobileNetV2, recognized for its lightweight and fast computation—the hybrid model achieves a strong balance between accuracy and computational efficiency. Each crop dataset underwent preprocessing and augmentation, followed by transfer learning-based training, enabling the model to adapt effectively to diverse disease types and environmental variations.

The experimental outcomes revealed that the model consistently achieved high accuracy across all datasets (ranging from 91% to 97%), with correspondingly high precision and recall values. The results validate that the hybrid design generalizes well and outperforms conventional CNN-based models such as AlexNet, ShuffleNet, and standalone EfficientNet variants.

Overall, this project demonstrates that the Hybrid EfficientNetB0–MobileNetV2 model can be effectively applied to multiple agricultural datasets independently, achieving reliable disease detection performance in each case. The work is done with separate datasets and recorded the values. So this lightweight design also enables deployment on mobile or edge devices, facilitating real-time field-based disease monitoring. Thus, the proposed system contributes to the advancement of precision agriculture, empowering farmers to make informed decisions, reduce crop loss, and promote sustainable farming practices.

CHAPTER 9

9. REFERENCES

1. A Hybrid AlexNet-ShuffleNet framework for plant leaf disease detection (Nasim Banu Shah, Springer,2025)
2. Sunil CK, Jaidhar CD, Patil N (2021) Cardamom plant disease detection approach using EfficientNetV2. IEEE Access 10:789–804
3. Shovon MSH, Mozumder SJ, Pal OK, Mridha MF, Asai N, Shin J (2023) PlantDet: a robust multi-model ensemble method based on deep learning for plant disease detection. IEEE Access 11:34846–34859
4. Hosny MK, El-Hady WM, Samy FM, Vrochidou E, Papakostas GA (2023) Multi-class classification of plant leaf diseases using feature fusion of deep convolutional neural network and local binary pattern. IEEE Access 11:62307–62317
5. Jiang P, Chen Y, Liu B, He D, Liang C (2019) Real-time detection of apple leaf diseases using deep learning approach based on improved convolutional neural networks. IEEE Access 7:59069–59080
6. Annabel LSP, Annapoorani T, Deepalakshmi P (2019) Machine learning for plant leaf disease detection and classification–a review. In: 2019 International conference on communication and signal processing (ICCSP), IEEE, pp 0538–0542
7. Amin H, Darwish A, Hassanien AE, Soliman M (2022) End-to-end deep learning model for corn leaf disease classification. IEEE Access 10:31103–31115
8. Sajid M, Ali N, Dar SH, Ratyal NI, Butt AR, Zafar B, Shafique T, Baig MJA, Riaz I, Baig S (2018) Research article data augmentation-assisted makeup-invariant face recognition. Math Probl Eng 2018:1–10
9. Xiao Z, Shi Y, Zhu G, Xiong J, Wu J (2023) Leaf disease detection based on lightweight deep residual network and attention mechanism. IEEE Access 11:48248–48258
10. Alnowami M, Taha E, Alsebaeai S, Anwar SM, Alhawsawi A (2022) MR image normalization dilemma and the accuracy of brain tumor classification model. J Radiat Res Appl Sci 15(3):33–39
11. Jasim MA, AL-Tuwaijari JM (2020) Plant leaf diseases detection and classification using image processing and deep learning techniques. In: 2020 International conference on computer science and software engineering (CSASE), Duhok, Iraq, pp 259–265
12. Strisciuglio N, Lopez-Antequera M, Petkov N (2020) Enhanced robustness of convolutional networks with a push–pull inhibition layer. Neural Comput Appl 32:17957–17971

13. Zhang X, Zhou X, Lin M, Sun J (2018) Shufflenet: an extremely efficient convolutional neural network for mobile devices. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 6848–6856
14. Vishnoi VK, Kumar K, Kumar B, Mohan S, Khan AA (2022) Detection of apple plant diseases using leaf images through convolutional neural network. IEEE Access 11:6594–6609
15. Saponara S, Elhanashi A (2021) Impact of image resizing on deep learning detectors for training time and model performance. In: International conference on applications in electronics pervading industry, environment and society. Springer International Publishing, Cham, pp 10–17
16. Najafabadi FS, Sadeghi MT (2022) AgriNet: a new classifying convolutional neural network for detecting agricultural products' diseases. J AI Data Min 10(2):285–302
17. Rice dataset. https://www.kaggle.com/datasets/vbookshelf/rice-leaf-diseases. Accessed Sept 2023
18. Sugarcane dataset. https://www.kaggle.com/datasets/prabhakaransoundar/sugarcane-disease-dataset. Accessed Sept 2023
19. Blackgram Plant Leaf Disease Dataset. https://data.mendeley.com/datasets/zfcv9fmrgv/3. Accessed Sept 2023
20. Dataset of Tomato Leaves. https://data.mendeley.com/datasets/ngdgg79rzb/1. Accessed Sept 2023

APPENDIX A- SAMPLE CODE :

Preprocessing Code:

```
# BLACKGRAM DATA PREPROCESSING SCRIPT

from google.colab import drive

drive.mount('/content/drive', force_remount=True)

import os

import shutil

import cv2

import numpy as np

from sklearn.model_selection import train_test_split

from tqdm import tqdm

BASE_PATH = '/content/drive/MyDrive/CapStoneProjectDataSets'

INPUT_PATH = f'{BASE_PATH}/Blackgram'

OUTPUT_PATH = f'{BASE_PATH}/Blackgram_split'  # Preprocessed data folder
```

```
IMG_SIZE = (224, 224)
TEST_RATIO = 0.2
# STEP 2: DELETE EXISTING PREPROCESSED DATA
# ============================================================
if os.path.exists(OUTPUT_PATH):
    shutil.rmtree(OUTPUT_PATH)
    print(f"🧹 Deleted existing folder: {OUTPUT_PATH}")
else:
    print(f"✅ No existing preprocessed folder found at: {OUTPUT_PATH}")
os.makedirs(f"{OUTPUT_PATH}/train", exist_ok=True)
os.makedirs(f"{OUTPUT_PATH}/test", exist_ok=True)
classes = [d for d in os.listdir(INPUT_PATH)
           if os.path.isdir(os.path.join(INPUT_PATH, d))
           and d.lower() not in ['blackgram_split', 'preprocessed_data']]
print("📁 Found Classes:", classes)
for cls in classes:
    cls_path = os.path.join(INPUT_PATH, cls)
    images = [os.path.join(cls_path, f) for f in os.listdir(cls_path)
              if f.lower().endswith(('.jpg', '.jpeg', '.png'))]
    print(f"\n🧾 Processing Class: {cls} ({len(images)} images)")
    if len(images) == 0:
        print(f"⚠️ No images found in {cls}, skipping...")
        continue
    train_imgs, test_imgs = train_test_split(images, test_size=TEST_RATIO,
random_state=42)
    for split_name, split_data in zip(['train', 'test'], [train_imgs, test_imgs]):
        out_dir = os.path.join(OUTPUT_PATH, split_name, cls)
        os.makedirs(out_dir, exist_ok=True)
        for img_path in tqdm(split_data, desc=f"{split_name.capitalize()} - {cls}"):
```

```
        try:
            img = cv2.imread(img_path)
            if img is None:
                continue
            img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
            img = cv2.resize(img, IMG_SIZE)
            cv2.imwrite(os.path.join(out_dir, os.path.basename(img_path)),
                        cv2.cvtColor(img, cv2.COLOR_RGB2BGR))
        except Exception as e:
            print(f"❌ Error processing {img_path}: {e}")
print("\n✅ Preprocessing Complete!")
print(f"📦 Data saved to: {OUTPUT_PATH}")
```

Training Code:

```
from google.colab import drive
drive.mount('/content/drive', force_remount=True)
import os
import numpy as np
import tensorflow as tf
from tensorflow.keras.applications import EfficientNetB0, MobileNetV2
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Dropout, GlobalAveragePooling2D, Concatenate,
Input, BatchNormalization, LeakyReLU
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint,
LearningRateScheduler
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score
import matplotlib.pyplot as plt
import math
BASE_PATH = "/content/drive/MyDrive/CapStoneProjectDataSets/Blackgram_split"
TRAIN_DIR = f"{BASE_PATH}/train"
TEST_DIR  = f"{BASE_PATH}/test"


IMG_SIZE = (224, 224)
BATCH_SIZE = 16
EPOCHS = 100
INIT_LR = 1e-4
best_model_path  =
"/content/drive/MyDrive/CapStoneProjectDataSets/blackgram_best_model.keras"
final_model_path =
"/content/drive/MyDrive/CapStoneProjectDataSets/blackgram_final_model_100epochs.keras
"
for path in [best_model_path, final_model_path]:
    if os.path.exists(path):
        os.remove(path)
        print(f"🧹 Deleted old model file: {path}")
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=45,
    width_shift_range=0.3,
    height_shift_range=0.3,
    shear_range=0.3,
    zoom_range=0.4,
    horizontal_flip=True,
    vertical_flip=True,
    brightness_range=[0.5, 1.5],
    channel_shift_range=20,
    fill_mode='nearest'
)
test_datagen = ImageDataGenerator(rescale=1./255)
train_gen = train_datagen.flow_from_directory(
    TRAIN_DIR,
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical'
)
test_gen = test_datagen.flow_from_directory(
    TEST_DIR,
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    shuffle=False
)
num_classes = len(train_gen.class_indices)
print("\n✅ Detected Classes:", train_gen.class_indices)
input_tensor = Input(shape=(IMG_SIZE[0], IMG_SIZE[1], 3))
# EfficientNetB0
effnet = EfficientNetB0(include_top=False, weights='imagenet', input_tensor=input_tensor)
for layer in effnet.layers[-80:]:  # unfreeze last 80 layers
    layer.trainable = True
effnet_feat = GlobalAveragePooling2D()(effnet.output)
# MobileNetV2
mobilenet = MobileNetV2(include_top=False, weights='imagenet',
input_tensor=input_tensor)
for layer in mobilenet.layers[-80:]:
    layer.trainable = True
mobilenet_feat = GlobalAveragePooling2D()(mobilenet.output)
```

```
# Combine both features
combined = Concatenate()([effnet_feat, mobilenet_feat])
x = Dense(1024)(combined)
x = BatchNormalization()(x)
x = LeakyReLU()(x)
x = Dropout(0.6)(x)
x = Dense(512)(x)
x = BatchNormalization()(x)
x = LeakyReLU()(x)
x = Dropout(0.5)(x)
x = Dense(256)(x)
x = BatchNormalization()(x)
x = LeakyReLU()(x)
x = Dropout(0.4)(x)
output = Dense(num_classes, activation='softmax')(x)
model = Model(inputs=input_tensor, outputs=output)
def cosine_annealing(epoch, total_epochs=EPOCHS, lr_max=INIT_LR, lr_min=1e-6):
    return lr_min + 0.5 * (lr_max - lr_min) * (1 + math.cos(math.pi * epoch / total_epochs))
lr_scheduler = LearningRateScheduler(cosine_annealing, verbose=1)
model.compile(
    optimizer=Adam(INIT_LR),
    loss=tf.keras.losses.CategoricalCrossentropy(label_smoothing=0.1),
    metrics=['accuracy']
)
callbacks = [
    lr_scheduler,
    EarlyStopping(monitor='val_accuracy', patience=12, restore_best_weights=True,
verbose=1),
    ModelCheckpoint(best_model_path, monitor='val_accuracy', save_best_only=True,
verbose=1)
]
print("\n🚀 Starting Optimized Training (100 Epochs)...")
history = model.fit(
    train_gen,
    validation_data=test_gen,
    epochs=EPOCHS,
    callbacks=callbacks,
    verbose=1
)
model.save(final_model_path)
print(f"\n✅ Final model saved at: {final_model_path}")
y_true = test_gen.classes
preds = model.predict(test_gen, verbose=0)
y_pred = np.argmax(preds, axis=1)
acc = accuracy_score(y_true, y_pred)
prec = precision_score(y_true, y_pred, average='macro', zero_division=0)
rec = recall_score(y_true, y_pred, average='macro', zero_division=0)
cm = confusion_matrix(y_true, y_pred)
# Specificity Calculation
spec_list = []
total = np.sum(cm)
for i in range(num_classes):
    TP = cm[i, i]
    FP = np.sum(cm[:, i]) - TP
    FN = np.sum(cm[i, :]) - TP
    TN = total - TP - FP - FN
    spec_list.append(TN / (TN + FP + 1e-7))
spec = np.mean(spec_list)
print("\n📊 === FINAL TEST PERFORMANCE METRICS ===")
print(f"✅ Accuracy: {acc * 100:.2f}%")
print(f"Precision: {prec * 100:.2f}%")
print(f"Recall (Sensitivity): {rec * 100:.2f}%")
print(f"Specificity: {spec * 100:.2f}%")

plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'], label='Train Acc', color='blue')
plt.plot(history.history['val_accuracy'], label='Val Acc', color='orange')
plt.title('Accuracy over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.grid()
plt.subplot(1,2,2)
plt.plot(history.history['loss'], label='Train Loss', color='green')
plt.plot(history.history['val_loss'], label='Val Loss', color='red')
plt.title('Loss over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid()
plt.show()
```

Testing Code:

```
from google.colab import drive
drive.mount('/content/drive', force_remount=True)
import os
import numpy as np

import matplotlib.pyplot as plt

from tensorflow.keras.models import load_model

from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img,
img_to_array

from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score,
ConfusionMatrixDisplay

BASE_PATH = "/content/drive/MyDrive/CapStoneProjectDataSets/Blackgram_split"

TEST_DIR = f"{BASE_PATH}/test"

MODEL_PATH =
"/content/drive/MyDrive/CapStoneProjectDataSets/blackgram_final_model_100epochs.keras
"

print("📦 Loading trained model...")

model = load_model(MODEL_PATH)

print(f"✅ Model loaded successfully from: {MODEL_PATH}")

IMG_SIZE = (224, 224)

BATCH_SIZE = 16

test_datagen = ImageDataGenerator(rescale=1./255)

test_gen = test_datagen.flow_from_directory(

    TEST_DIR,

    target_size=IMG_SIZE,

    batch_size=BATCH_SIZE,

    class_mode='categorical',

    shuffle=False

)

print("\n🔍 Evaluating model on test dataset...")

preds = model.predict(test_gen, verbose=1)

y_true = test_gen.classes

y_pred = np.argmax(preds, axis=1)

acc = accuracy_score(y_true, y_pred)
```

```
prec = precision_score(y_true, y_pred, average='macro', zero_division=0)
rec = recall_score(y_true, y_pred, average='macro', zero_division=0)
cm = confusion_matrix(y_true, y_pred)

# Specificity
num_classes = len(test_gen.class_indices)
spec_list = []
total = np.sum(cm)
for i in range(num_classes):
    TP = cm[i, i]
    FP = np.sum(cm[:, i]) - TP
    FN = np.sum(cm[i, :]) - TP
    TN = total - TP - FP - FN
    spec_list.append(TN / (TN + FP + 1e-7))
spec = np.mean(spec_list)
print("\n📊 === FINAL BLACKGRAM TEST PERFORMANCE METRICS ===")
print(f"✅ Accuracy: {acc * 100:.2f}%")
print(f"Precision: {prec * 100:.2f}%")
print(f"Recall (Sensitivity): {rec * 100:.2f}%")
print(f"Specificity: {spec * 100:.2f}%")
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=list(test_gen.class_indices.keys()))
disp.plot(cmap='Greens', xticks_rotation=45)
plt.title("Confusion Matrix - Blackgram Leaf Disease Classification")
plt.show()
print("\n🧾 Displaying one image from each disease class with confidence:")
class_labels = list(test_gen.class_indices.keys())
plt.figure(figsize=(15, 6))
```

```
for i, class_name in enumerate(class_labels):
    class_dir = os.path.join(TEST_DIR, class_name)
    sample_img = np.random.choice(os.listdir(class_dir))  # Random image from class
    img_path = os.path.join(class_dir, sample_img)
    # Load and preprocess image
    img = load_img(img_path, target_size=IMG_SIZE)
    img_array = img_to_array(img) / 255.0
    img_batch = np.expand_dims(img_array, axis=0)
    # Predict
    pred_probs = model.predict(img_batch, verbose=0)
    pred_class = np.argmax(pred_probs)
    confidence = np.max(pred_probs) * 100
    pred_label = class_labels[pred_class]
    # Display
    plt.subplot(1, len(class_labels), i + 1)
    plt.imshow(img)
    plt.axis('off')
    color = 'green' if pred_label == class_name else 'red'
    plt.title(f"T: {class_name}\nP: {pred_label}\n({confidence:.1f}%)", color=color,
fontsize=9)
plt.tight_layout()
plt.show()
```