Detection of Apple Plant Diseases Using An Efficient Ensemble Approach

Report submitted to the SASTRA Deemed to be University
in partial fulfillment of the requirements
for the award of the degree of

Bachelor of Technology

Submitted by

A B S S Kowshik
(Reg. No.: 125015182, Information Technology)
B Venkata Lakshman
(Reg. No.: 125004338, Electronics and Communication Engineering)
N Bhargav
(Reg. No.: 125157042, Computer Science & Engineering(Cyber Security and Blockchain Technology))

May 2025

SCHOOL OF COMPUTING

THANJAVUR, TAMIL NADU, INDIA – 613 401

SASTRA

ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION

DEEMED TO BE UNIVERSITY

(U/S 3 of the UGC Act,1956)

THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

THANJAVUR | KUMBAKONAM | CHENNAI

SCHOOL OF COMPUTING

THANJAVUR – 613 401

Bonafide Certificate

This is to certify that the project report titled "**Detection of Apple Plant Diseases Using An Efficient Ensemble Approach** " submitted in partial fulfillment of the requirements for the award of the degree of B. Tech. Electronics and Communication Engineering to the SASTRA Deemed to be University, is a bona-fide record of the work done by **Mr. Venkata Lakshman Bhavana** (**Reg. No. 12500448**) (**Mr. A B B S Kowshik** (**Reg. No. : 125015182, B.Tech Information Technology**) and **Mr. N Bhargav (Reg. No. : 125157042, B.Tech Computer Science & Engineering(Cyber Security and Blockchain Technology))** during the final semester of the academic year 2024- 25, in the **School of Computing**, under my supervision. This report has not formed the basis for the award of any degree, diploma, associateship, fellowship or other similar title to any candidate of any University.

Signature of Project Supervisor :

Name with Affiliation :

Date :

Project *Viva voce* held on _____

Examiner 1 **Examiner 2**

SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 of the UGC Act, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA
THANJAVUR | KUMBAKONAM | CHENNAI

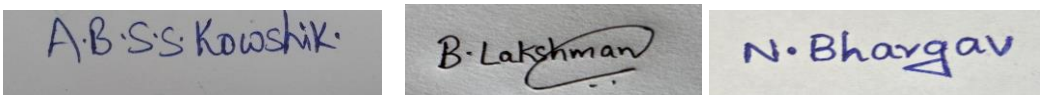
SCHOOL OF COMPUTING

THANJAVUR – 613 401

Declaration

We declare that the project report titled "Detection of Apple Plant Diseases Using An Efficient Ensemble Approach" submitted by me/us is an original work done by me/us under the guidance of **Dr. Gokila RG, Asst. Professor - II, School of Computing, SASTRA Deemed to be University** during the final semester of the academic year 2024-25, in the **School of Computing**. The work is original and wherever I/We have used materials from other sources, I/We have given due credit and cited them in the text of the report. This report has not formed the basis for the award of any degree, diploma, associate-ship, fellowship or other similar title to any candidate of any University.

Signature of the candidate(s) :



Name of the candidate(s) : A B S S Kowshik, B Venkata Lakshman, N Bhargav

Date : 05- 05- 2025

Acknowledgements

We would like to thank our Honorable Chancellor **Prof. R. Sethuraman** for providing us with an opportunity and the necessary infrastructure for carrying out this project as a part of our curriculum.

We would like to thank our Honorable Vice-Chancellor **Dr. S.Vaidhyasubramaniam** and **Dr. S. Swaminathan,** Dean**,** Planning & Development, for the encouragement and strategic support at every step of our college life.

We extend our sincere thanks to **Dr. R. Chandramouli,** Registrar, SASTRA Deemed to be University for providing the opportunity to pursue this project.

We extend our heartfelt thanks to **Dr. V. S. Shankar Sriram**, Dean, School of Computing, **Dr. R. Muthaiah**, Associate Dean, Research, **Dr. K.Ramkumar**, Associate Dean, Academics, **Dr. D. Manivannan**, Associate Dean, Infrastructure, **Dr. R. Alageswaran**, Associate Dean, Students Welfare.

Our guide Dr. Gokila RG, Asst. Professor - II, School of Computing was the driving force behind this whole idea from the start. Her deep insight in the field and invaluable suggestions helped us in making progress throughout our project work. We also thank the project review panel members for their valuable comments and insights which made this project better.

We would like to extend our gratitude to all the teaching and non-teaching faculties of the School of Computing who have either directly or indirectly helped us in the completion of the project.

We gratefully acknowledge all the contributions and encouragement from my family and friends resulting in the successful completion of this project. We thank you all for providing me an opportunity to showcase my skills through projects.

Table of Contents

Title	Page No.
Bona-fide Certificate	ii
Declaration	iii
Acknowledgements	iv
List of Figures	vii
List of Tables	viii
Abbreviations	ix
Abstract	x
1. Introduction	
1.1. Introduction	1
1.2. HighLights of Dataset	1
1.3. Limitations of Traditional Approaches	2
1.4. Literature Survey	2
1.5. Motivation	6
2. Objectives	7
3. Experimental Work / Methodology	
3.1. Methodology	8
3.2. Data Preprocessing	8
3.3. Splitting the data	9
3.4. VGG Net for leaf disease Detection	9
3.5. ResNet50 for leaf disease Detection	10
3.6. ResNet512 for leaf disease Detection	12
3.7. Inception V3 for leaf disease Detection	13
3.8. Xception for leaf disease Detection	14
3.9. DenseNet for leaf disease Detection	15
3.10. MobileNetV2 for leaf disease Detection	16
3.11. Bagging an ensemble method for leaf disease Detection	17

3.12. Boosting an ensemble method for leaf disease Detection 18
3.13. Stacking an ensemble method for leaf disease Detection 19
3.14. Voting an ensemble method for leaf disease Detection 21
3.15. Blending an ensemble method for leaf disease Detection 23
4. Results and Discussion 26
5. Conclusions and Further Work 38
6. References 39
7. Appendix 40
7.1 Sample Source code

List of Figures

Figure No.	Title	Page No.
4.1	Images from each class	26
4.2	Loss and Accuracy Curves for VGGNet	26
4.3	Testing and Training Accuracies for VGGNet	26
4.4	Loss and Accuracy Curves for ResNet50	27
4.5	Testing and Training Accuracies for ResNet50	27
4.6	Loss and Accuracy Curves for ResNet512	27
4.7	Testing and Training Accuracies for ResNet512	28
4.8	Loss and Accuracy Curves for Inception	28
4.9	Testing and Training Accuracies for Inception	28
4.10	Loss and Accuracy Curves for Xception	29
4.11	Testing and Training Accuracies for Xception	29
4.12	Loss and Accuracy Curves for DenseNet	29
4.13	Testing and Training Accuracies for DenseNet	30
4.14	Loss and Accuracy Curves for MobileNet	30
4.15	Testing and Training Accuracies for MobileNet	30
4.16	Confusion Matrix	31
4.17	Classification Reports	31
4.20	Loss and Accuracy Curves for Bagging	33

4.21	Classification Report for Bagging	33
4.22	Loss and Accuracy Curves for Boosting	34
4.23	Classification Report for Boosting	34
4.24	Loss and Accuracy Curves for stacking	35
4.25	Classification Report for Stacking	36
4.26	Loss and Accuracy Curves for Voting	36
4.27	Classification Report for Voting	36
4.28	Loss and Accuracy Curves for Blending	37
4.29	Classification Report for Blending	37

List of Tables

Table No.	**Table name**	**Page No.**
1.2	Plant Village Image Dataset	2
4.18	Comparison Table for CNN Models	32
4.19	Comparison Table for Ensembling Models	32

Abbreviations

CNN	Convolutional neural network
HSV	Hue Saturation Value
ReLU	Rectified Linear Unit
VGG	Visual Geometry Group
ResNet	Residual Network
YOLO	You Only Look Once
IOT	Internet of Things
DL	Deep Learning
ROC	Receiver Operating Characteristic
GPU	Graphics Processing Unit
ML	Machine Learning
F1-Score	Harmonic Mean of Precision and Recall

Abstract

Apple Plant diseases pose a significant threat to agricultural productivity, often leading to considerable crop losses globally. Early detection and identification of these diseases ensure healthy yields and reduce losses due to economic impacts. Existing Deep Learning methods pose challenges due to their reliance on large labelled datasets and the high computational costs associated with their implementation. Proposed work aims to develop a practical and efficient method for Apple plant disease detection by combining ensembling methods, data augmentation, and lightweight CNN designs. An ensembling based approach integrating other computational techniques to provide a robust and scalable solution for disease identification. Lightweight architectures along with augmentation techniques such as shifts, shears, scaling, zooming, and flipping were applied to the dataset to counter the limitations in the dataset and reduce computational complexity. The publicly available PlantVillage dataset will be used to train the ensemble model to classify diseases in apple leaves. This approach ensures high performance with minimal computational overhead, addressing the limitations of existing deep learning models and enhancing scalability for effective agricultural disease management.

References: **https://ieeexplore.ieee.org/document/10002365**

CHAPTER 1

SUMMARY OF THE BASE PAPER

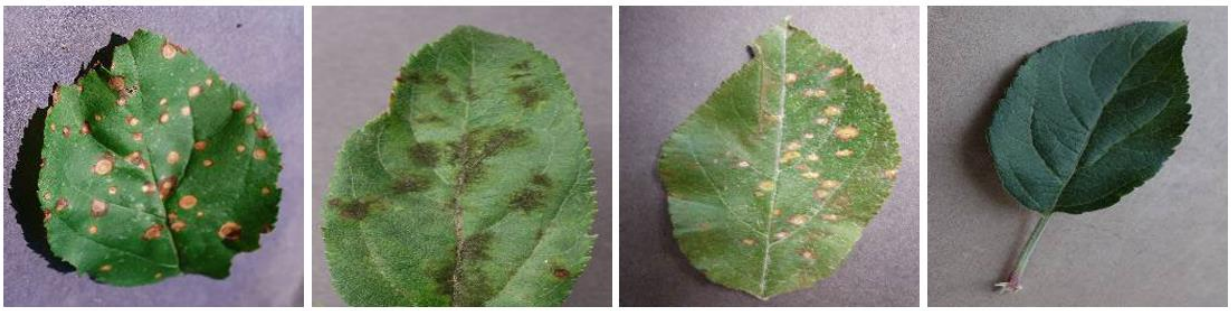
Title	**:**	Detection of Apple Plant Diseases Using An Efficient Ensemble Approach
Publisher	**:**	IEEE
Year	**:**	2022
Journal Name	**:**	Detection of Apple Plant Diseases Using Leaf Images Through Convolutional Neural Network
Base paper URL	**:**	https://ieeexplore.ieee.org/document/10002365

1.1 INTRODUCTION

Apple leaf disease has an adverse effect on farm productivity, driven by factors such as unseasonal weather, poor irrigation, and disease epidemics. Applications of recent technology, including intelligent monitoring networks and precision agriculture equipment, are transforming apple cultivation by enhancing disease management and lowering its costs. In apple cultivation, neural networks are also important in the detection of diseases, yield prediction, and quality determination. In particular, deep learning approaches employing ensemble methods are effective to classify apple leaf diseases with high accuracy, although limitations arise owing to diversity constraints in datasets. Utilization of the PlantVillage dataset facilitates training of models but customized datasets remain necessary for real-world accuracy. Further, fine-tuning hyperparameters is important to develop efficient and dependable diagnostic models for identifying apple leaf diseases.

1.2 HIGHLIGHTS OF THE DATASET

The apple leaf pictures are sampled from the PlantVillage dataset available publicly .The dataset holds 3171 RGB apple plant leaf images split across four classes. The three classes relate to 3 diseases of apples such as black rot, scab, and apple cedar rust. The fourth class stands for healthy/uninfected leaf images. The carefully annotated apple leaf pictures of dimension 256x256 for all four categories were taken with a simple background at different stages of plant development in laboratory conditions . Figure 1 presents sample images from each class. The classes for the disease are based on the apple diseases and the healthy class represents leaves free from any disease. Plantvillage dataset has been employed in a high number of research works.



(a) Black Rot (b) Apple Scab (c) Cedar Apple Rust (d) Healthy leaf

Figure 1.1

Infection	Number of Images
Black Rot	621
Scab	630
Cedar Rust	275
Healthy Leaf	1645

Table 1.2

1.3. Limitations of Traditional Approaches

Conventionally, plant disease detection has been based on visual inspection by agricultural specialists or simple machine learning models like Decision Trees, K-Nearest Neighbors (KNN), and Support Vector Machines (SVM), employing hand-engineered features from leaf images. Although these methods have shown preliminary assistance in disease detection, they tend to fail in identifying the complex patterns, textures, and color gradations involved in various plant diseases. Further, feature extraction manually is time-consuming and prone to variability, which results in compromised reliability and scalability in practical agricultural applications. These constraints have brought to the forefront the need for more efficient and automated systems, particularly with the increasing diversity and complexity of plant diseases. In response to this need, deep learning methods have emerged, with Convolutional Neural Networks (CNNs) leading the charge in revolutionizing image analysis by learning hierarchical representations directly from raw image data without relying on manual features. CNNs have shown phenomenal success in a wide range of computer vision applications, including disease detection in plants.

1.4. Literature Survey

The challenge of early and accurate detection of plant diseases, particularly in apple cultivation, has garnered significant research attention in recent years. With the advent of deep learning technologies and the availability of image datasets such as PlantVillage, a variety of CNN-based and hybrid methods have emerged. This section offers a comprehensive survey of pivotal works that have shaped the field of automated plant disease detection.

- **"Detection of Apple Plant Diseases Using Leaf Images Through Convolutional Neural Network"** by Vibhor Kumar Vishnoi, Brajesh Kumar, Krishna Kumar, Shashank Mohan, and Arfat Ahmad Khan (2022)**.**

This study introduces an effective method for the detection of apple plant diseases using Convolutional Neural Networks (CNNs) applied to leaf image datasets. The authors propose a deep learning-based solution that automates the disease identification process using CNN models trained on the PlantVillage dataset. By eliminating the need for manual inspection, the approach significantly accelerates disease diagnosis, offering real-time results and facilitating early intervention. The CNN model is designed to identify disease-specific visual patterns and categorize leaves into healthy or diseased classes, ensuring non-invasive and scalable disease detection.

Despite its high accuracy, the model's performance is heavily dependent on the availability of a large, well-labeled image dataset. Moreover, the system may face difficulties in adapting to real-world variability, such as different lighting conditions, leaf orientations, and camera resolutions. Environmental inconsistencies often cause performance degradation, highlighting the importance of data diversity in training. Additionally, training and deploying deep CNN models require substantial computational power, which may limit the applicability of such solutions in resource-constrained agricultural environments.

- **"Identification of Plant Leaf Diseases by Deep Learning Based on Channel Attention and Channel Pruning"** by Riyao Chen and Haixia Qi (2022).

  This paper presents a refined deep learning model that leverages channel attention mechanisms and channel pruning to improve the efficiency and accuracy of plant disease detection. Channel attention helps the network focus on the most relevant features, while channel pruning eliminates less important parameters, reducing the model's complexity. The proposed model demonstrates an accuracy of 95.7% on the PlantVillage dataset while reducing the model size by nearly 58%. This makes it a lightweight, high-performance candidate for real-time disease identification, particularly suitable for embedded devices and mobile applications.

  However, the model's success depends on precise calibration of the pruning mechanism; poor tuning can lead to substantial accuracy losses. Moreover, transferring such a specialized model to different datasets or plant species can be challenging due to the need for fine-grained optimization and retraining. While the approach is promising for deployment in efficient edge-based systems, it demands careful trade-offs between performance and model simplicity.

- **"Automatic and Reliable Leaf Disease Detection Using Deep Learning Techniques"** by M. Shorten and T.M. Khoshgoftaar (2021)**.**

  This study provides a broad overview of recent advances in deep learning for plant disease detection, with particular attention to the emerging role of Vision Transformers (ViTs).

ViTs are shown to outperform CNNs in certain large-scale tasks due to their ability to capture global dependencies in image data. The paper highlights that ViTs, when trained on large annotated datasets, achieve impressive accuracy and robustness, especially for complex classification tasks involving multiple disease types and plant species.

However, ViTs come with significant trade-offs. The models are computationally intensive, requiring substantial GPU resources and high memory, which limits their practical application in agriculture where such infrastructure is often unavailable. Additionally, they require massive training datasets, and may struggle with domain-specific generalization when trained on generic plant datasets but deployed in new, uncontrolled field environments.

- **"Using Deep Learning for Image-Based Plant Disease Detection"** by S.P. Mohanty, D.P. Hughes, and M. Salathé (2016).

  As one of the pioneering works in the application of deep learning to plant pathology, this paper explores the use of deep CNNs to classify 26 diseases across 14 crop species using leaf images. The model was trained on the PlantVillage dataset and achieved high levels of classification accuracy, demonstrating the feasibility of using CNNs for automated crop disease identification. The work also inspired a wave of follow-up studies that investigated CNNs in other crop types, disease types, and image modalities.

  Despite the promising results, the authors note limitations in generalization. The model performed well on lab-quality images but struggled with images taken in natural settings with variable lighting, background noise, and occlusions. This reveals a gap between experimental conditions and real-world deployment. Future work needs to address domain adaptation, environmental noise, and integration with field-deployable tools such as smartphones or drones.

- **"A Deep Learning Approach for Detection and Localization of Leaf Anomalies"** by Davide Calabrò et al. (2022).

  This study diverges from typical supervised approaches by focusing on unsupervised anomaly detection using advanced autoencoder models such as Convolutional Autoencoders (CAE), Conditional Variational Autoencoders (CVAE), and Vector Quantized VAEs (VQ-VAE). These models are designed to reconstruct healthy leaf images, and deviations in reconstruction highlight anomalies indicative of disease. The approach is especially useful in detecting unknown diseases where labeled training data is unavailable, making it valuable in early outbreak scenarios or novel pathogen detection.

  While unsupervised learning offers flexibility and adaptability, it often requires extensive data and careful threshold tuning. Localization accuracy may be lower compared to supervised methods, and the interpretability of anomaly scores can be challenging for end-users. Additionally, deploying such systems in the field demands robust preprocessing

pipelines and high-quality thermal or RGB imagery to ensure meaningful anomaly detection.

- **"Deep Learning for Plant Identification and Disease Classification from Leaf Images"** by Jianping Yao et al. (2023).

  This paper proposes GSMo-CNN (Generalized Stacking Multi-Output Convolutional Neural Network), a complex architecture tailored for multi-task learning in plant disease diagnosis. The model is capable of performing multiple predictions—such as species identification and disease classification—simultaneously, which streamlines the diagnostic pipeline. Tested across multiple benchmark datasets, the GSMo-CNN outperforms conventional models in both accuracy and efficiency, setting a new standard for plant image classification tasks.

  However, the sophistication of this architecture comes at the cost of high computational demand. Training and tuning require advanced hardware and a deep understanding of model internals. Its deployment in low-resource agricultural environments is currently impractical without major simplifications or model compression techniques. Nevertheless, the paper establishes a strong case for exploring multi-output learning models in future agricultural AI systems.

- **"Data Augmentation for Improving Deep Learning in Image Classification Problem"** by M. Shorten and T.M. Khoshgoftaar (2021).

  This work emphasizes the role of data augmentation in enhancing the generalization and accuracy of deep learning models, especially when working with limited datasets. Techniques such as rotation, flipping, cropping, and color jittering artificially inflate the size of training datasets, helping models learn more robust features. The authors demonstrate that properly applied augmentations can significantly reduce overfitting and improve classification results across multiple vision tasks, including plant disease detection.

  Nevertheless, augmentation is not a panacea. Careless application of transformations may introduce image distortions or irrelevant features, leading to degraded model performance. Moreover, augmentation cannot eliminate biases present in the original data—if the dataset is skewed or lacks representative samples, the model may still inherit those biases. The paper suggests that augmentation should be used in combination with domain-specific preprocessing and balanced data collection efforts.

- **"Deep Learning Approaches for Image Recognition and Classification"** by S. S. S. S. Krishnan et al. (2022).

This paper presents a comprehensive review of deep learning models, with a strong focus on Convolutional Neural Networks (CNNs) as the dominant architecture for image recognition tasks. CNNs are praised for their ability to learn spatial hierarchies of features and their wide application across domains including agriculture, healthcare, and autonomous systems. The study also explores transfer learning as a strategy to reduce training time by reusing pre-trained models from large datasets like ImageNet.

However, transfer learning poses challenges when the target domain differs significantly from the source domain. For instance, features learned on general object datasets may not translate well to specific plant leaf patterns. In such cases, fine-tuning becomes essential, which requires additional labeled data and careful optimization. The review concludes that while CNNs remain powerful, their performance is heavily tied to data quality, domain similarity, and training infrastructure.

MERITS AND DEMERITS

Merits:

- Examination of advanced deep learning architectures like ResNet, Xception, DenseNet, and MobileNetV2 to improve plant disease detection through robust feature extraction capabilities.
- Integration of ensemble learning techniques such as bagging, boosting, stacking, voting, and blending using sequential models to enhance classification accuracy and model reliability.
- Emphasis on early detection of plant diseases to improve crop yield, reduce economic losses, and promote sustainable agricultural practices.
- Use of standard evaluation techniques like confusion matrix, precision, recall, and F1-score, demonstrating practical validation and effectiveness on benchmark datasets like PlantVillage.

Demerits:

- High dependency on large volumes of labeled data for training, which limits model applicability in real-world agricultural environments with scarce data.
- High computational resource demands and longer training times due to the complexity of deep learning and ensemble learning approaches.
- Potential overfitting risks associated with ensemble methods like stacking and blending if hyperparameter tuning is not managed properly.

1.5 MOTIVATION

The increasing impact of apple plant diseases on agricultural productivity has underscored the need for early and accurate detection methods. Traditional approaches, which depend on manual inspection by experts, are time-consuming, costly, and often inconsistent, particularly when applied across large-scale farming environments. Motivated by these limitations, this project seeks to harness the power of deep learning and image-based analysis to build a robust, automated disease detection system.

Existing deep learning solutions show promise, but they often struggle with key challenges: high dependency on large, labeled datasets, the need for powerful computational resources, and the risk of overfitting due to dataset limitations. The motivation behind this work is to address these challenges by developing an efficient and scalable model that employs ensemble learning strategies combined with lightweight CNN architectures. By using the publicly available PlantVillage dataset and enhancing it through data augmentation techniques (such as flipping, scaling, and shearing), the model aims to generalize better and improve real-world performance.

Furthermore, this project aspires to support precision agriculture by enabling timely diagnosis of diseases like black rot, scab, and cedar rust. By integrating various ensemble methods—such as bagging, boosting, voting, and stacking—alongside color-based segmentation, the system enhances classification accuracy while minimizing computational overhead. Ultimately, the goal is to provide farmers and agricultural stakeholders with a practical, low-cost solution that improves crop yield, reduces pesticide use, and supports sustainable farming practices.

CHAPTER 2

OBJECTIVES

• To develop an automated system for early detection of apple plant diseases using advanced image-based analysis by leveraging state-of-the-art deep learning models and ensemble techniques, enabling timely interventions and minimizing crop losses caused by undetected or late-detected infections.

• To overcome limitations of traditional disease detection methods, such as the high time consumption, labor-intensive manual inspection, and subjective human errors, by creating a scalable, objective, and efficient deep learning-driven pipeline that ensures consistent results under diverse agricultural conditions.

• To utilize data augmentation techniques including horizontal/vertical flipping, random shifting, shearing, scaling, and zooming to synthetically expand the dataset, improve generalization, prevent model overfitting due to limited annotated image samples, and ensure the deep learning model performs well on unseen data.

• To implement and compare multiple convolutional neural network (CNN) architectures, such as VGGNet, ResNet-50/152, InceptionV3, Xception, DenseNet201, and MobileNetV2, to identify the most accurate and computationally efficient model for robust classification of apple leaf diseases across various severity levels and visual patterns.

• To apply ensemble learning strategies, including Bagging (e.g., Random Forest), Boosting (e.g., XGBoost, AdaBoost), Stacking, Voting, and Blending, to aggregate predictions from multiple models, thereby enhancing prediction stability, improving classification performance, and reducing the variance and bias of individual models

• To minimize computational overhead and resource usage by optimizing model architecture, using lightweight models where appropriate, and adopting inference-efficient techniques, ensuring the final solution is practical for deployment in real-time scenarios, including mobile or edge devices in remote agricultural environments.

• To support precision agriculture and sustainable farming practices by providing a reliable, real-time diagnostic tool that empowers farmers with immediate, actionable insights, reduces dependency on chemical pesticides, improves yield quality, and contributes to environmentally responsible agriculture.

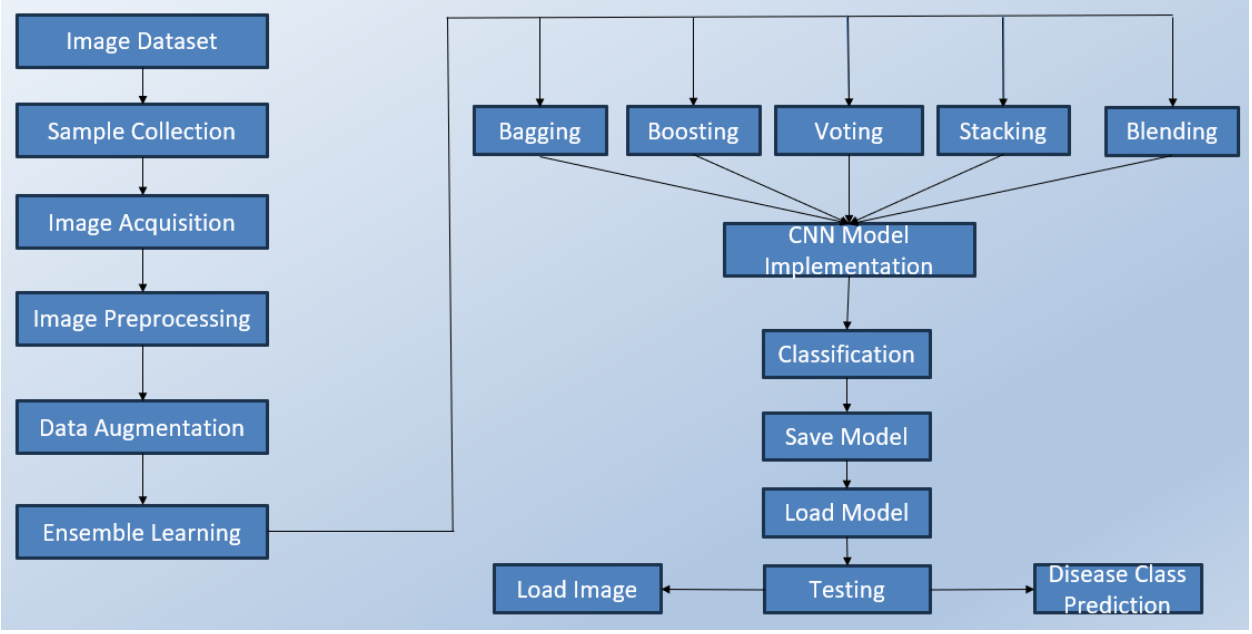
CHAPTER 3

EXPERIMENTAL WORK / METHODOLOGY

3.1. Methodology

The detection system follows a pipeline approach:

- Data Acquisition – Collect labeled apple leaf images from the PlantVillage dataset.
- Preprocessing and Augmentation – Prepare images using techniques like resizing, normalization, and augmentation.
- Model Building – Use transfer learning on various CNN architectures (VGGNet, ResNet, InceptionV3, etc.).
- Ensemble Learning – Combine multiple model outputs using techniques like Bagging, Boosting, Voting, Stacking, and Blending.
- Evaluation – Assess performance using accuracy, precision, recall, F1-score, and confusion matrices.



3.2. Data Preprocessing

Data preprocessing in CNN deep learning involves preparing and organizing the data before feeding it into the neural network for training. This step is crucial for ensuring that the network can learn effectively from the data and produce accurate predictions. Here's what it typically involves:

a. Data Collection
b. Data Shuffling

c. Data Augmentation & Normalization
d. Image Resizing
e. Image Resizing

a. Data Collection:
Data collection for Apple Leaf Disease Detection involves gathering a large number of apple leaf images showing various disease conditions. We use the PlantVillage dataset, which contains 3171 images categorized into different classes such as Apple Scab, Black Rot, Cedar Apple Rust, and healthy leaves, enabling accurate training and testing of deep learning models.

b. Data Shuffling: Data shuffling in CNN training randomizes the dataset to prevent bias and overfitting, fostering better feature diversity and generalization. This technique, executed at the start of each epoch, diversifies exposure, preventing sequence memorization.

c. Data Augmentation: We did Data augmentation to our dataset by generating variations through transformations like brightness, contrast, sharpness, standard size, rotating, zooming, flipping, cropping etc. This enriches the dataset, aiding CNN models in better generalization and accurate Expression detection, mitigating overfitting. Before and after applying augmentation. Data augmentation efficiently expands the size and diversity of the training dataset by applying these transformations to the training images. This enhances the ability of machine learning models to perform better on tasks like image classification, object detection and segmentation as well as to better generalize to new data. Applying data augmentation wisely is crucial, though, taking into account the needs of the machine learning task and the features of the dataset.

d. Normalization: Normalization in CNN-based classification scales pixel values to a standardized range (e.g., 0-1), ensuring uniformity and faster model convergence. This enhances stability, efficiency, and accuracy by mitigating variations in pixel intensity.

e. Image Resizing: We are performing Image resizing standardized dimensions from 512x512 to 128x128 pixels, maintaining 3 color channels(RGB), aiding CNN model efficiency.

3.3. Splitting the data

The Next step in our project is splitting the datasets into training, validation and testing sets. The training set is used to train the model, the validation set is used to set hyperparameters and monitor performance during training, and the test set is used to evaluate the performance of the final model. We are using the flow_from_directory which is the class in the keras. When working with picture data for deep learning tasks, it is frequently utilized, especially for tasks like image categorization and object detection.
Arranging Picture Data: Initially, arrange your picture data into folders. A class is represented by each directory, which contains the photos for that class. As an illustration, suppose you had two directories, each containing pictures of either Blackrot or Apple scab, and your assignment

involved classification between two classes (for instance, " Black Rot " and " Apple scab"). Configuring ImageDataGenerator Next, you make an instance of the class ImageDataGenerator. A variety of preprocessing and augmentation techniques for picture data are covered in this class.To generate batches, utilize the ImageDataGenerator instance's flow_from_directory method.

3.4. VGGNet for Leaf Disease Detection

This Proposed deep learning model classifies the plant diseases using the PlantVillage dataset images. The dataset is distributed over subdirectories, one for each disease class or for healthy leaf, which is supported by Keras's flow_from_directory for enhanced extraction and labeling. The images are all resized to 224x224 pixels and normalized to the range [0, 1] pixels. To enhance generalization and prevent overfitting, the model makes use of data augmentation techniques including random rotation of up to 40°, zoom of up to 50%, shear of up to 30%, and horizontal flip. Image data are split into 80% for training and 20% validation using the validation_split parameter.

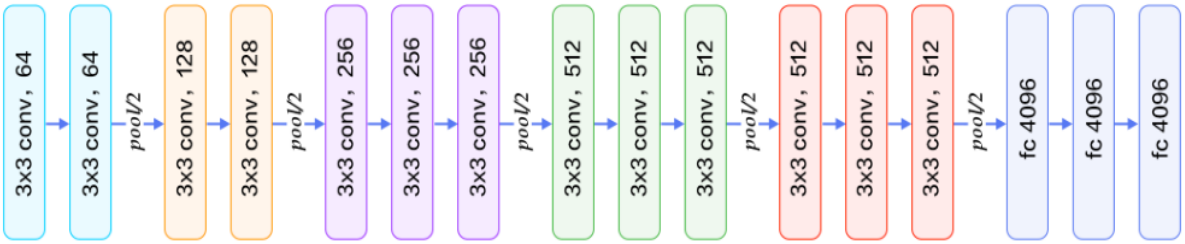
The model uses the VGG16 structure, which is a proven convolutional neural network, with the top layers eliminated and the pretrained ImageNet weights. The input shape is defined as (224, 224, 3). The first 15 layers of VGG16 are frozen to preserve basic visual features, while the remaining layers are used to learn disease-specific patterns.

A custom classification head is added with Keras's Sequential API. The base model output is provided via a GlobalAveragePooling2D layer to reduce tensor sizes while preserving spatial data. Next, a BatchNormalization layer to speed up training and a Dense layer with 256 units and ReLU activation. A Dropout layer (rate 0.5) for regularization. Finally, a `Dense` layer with softmax activation gives class probabilities for the count of plant disease classes.

The model is trained using Adam optimizer with a learning rate of 0.0001. Loss function used is categorical cross entropy, which is suitable for multi-class classification. Two callbacks are included: EarlyStopping terminates training if validation loss fails to improve in five epochs and saves the best weights, and ReduceLROnPlateau decreases learning rate if validation loss plateaus, by fine-tuning.

Training is for 10 epochs, and validation set performance is observed after each epoch. A function called plot_training () , plots training and validation accuracy and loss and marks the best epoch to identify the overfitting and underfitting iterations

Finally, the model is validated on the validation set using `model. evaluate () ` to determine final accuracy and loss, measuring performance on unseen data. The VGG16-based pipeline seamlessly integrates transfer learning, data augmentation, regularization, and interpretability for plant disease prediction based on the health of the input leaf.



VGGNET Architecture

3.5. ResNet50 for Leaf Disease Detection

The process begins with importing necessary libraries such as `numpy`, `matplotlib. pyplot` for visualization, and TensorFlow's Keras API for deep learning. Keras layers such as `Dense`, `Dropout`, `BatchNormalization`, and `GlobalAveragePooling2D` are used to build a robust classification model. Optimization is handled by the Adam optimizer, and callbacks like `EarlyStopping` and `ReduceLROnPlateau` are used for effective training. Scikit-learn's `classification_report` is utilized to measure performance post-prediction.

The data is processed using `ImageDataGenerator`, and pixel values are scaled while real-time augmentations like rotation, zoom, shear, and horizontal flipping are included. The generalization of the model is improved by these augmentations. 80% training and 20% validation datasets are split using `validation_split`. Images are resized to 224×224 pixels and loaded in batches of 32 using `flow_from_directory`.

Model architecture is adopted using Keras's Sequential API. ResNet50V2 is loaded without the top layer included (`include_top=False`) and with ImageNet weights. Base model's layers are frozen to preserve the pre-trained features.

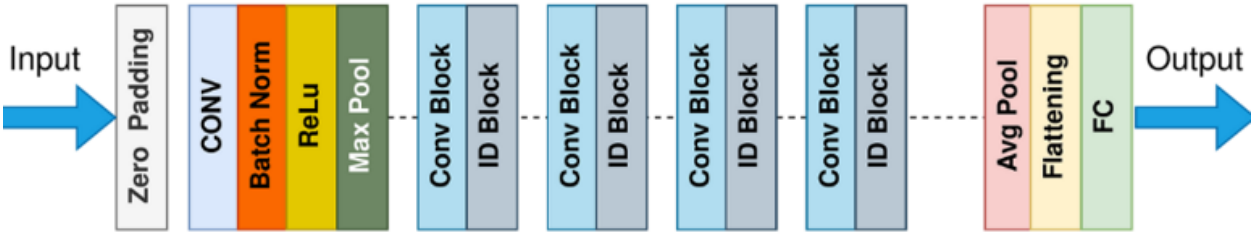
A `GlobalAveragePooling2D` reduces feature maps, and `BatchNormalization` stabilizes the training. A dense layer with 512 units of ReLU-activated units adds non-linearity, and a `Dropout` layer with rate 0.5 prevents overfitting. A final `Dense` with softmax activation provides class probabilities for the 38 classes.

Training is done using the Adam optimizer (learning rate 0.0001) and categorical cross-entropy loss, suitable for multi-class classification. Evaluation metric is accuracy. Training is done for 10 epochs with `EarlyStopping` to halt training when validation loss stops improving, and `ReduceLROnPlateau` to reduce learning rate when training plateaus as demonstrated in graphs.

The model is then tested on the validation set after being trained with `evaluate`, providing loss and accuracy. Prediction is performed using `model. predict`, and class labels are obtained using `argmax`. The results are measured using `classification_report`, providing precision, recall, and F1-score for all of them.

Finally, the trained model is saved in HDF5 format with a naming scheme including model name, dataset, and last accuracy. The weights are also saved independently for future reuse during inference or fine-tuning.

To sum up, the pipeline presented herein provides a solution for plant disease classification through ResNet50V2 with emphasis on model reliability, accuracy.



ResNet50 Architecture

3.6. ResNet512 for Leaf Disease Detection

Importing required libraries like numpy, matplotlib.pyplot for plotting, and TensorFlow's Keras API for deep learning starts the process. Keras layers like Dense, Dropout, BatchNormalization, and GlobalAveragePooling2D are utilized to build a high-capacity classification model. Optimization is taken care of by the Adam optimizer, and callbacks like EarlyStopping and ReduceLROnPlateau are used for stable and efficient training. Scikit-learn's classification_report is used to evaluate model performance after prediction.

ImageDataGenerator is used to process the dataset with pixel values standardized to between 0 and 1. Data augmentations such as rotation, shear, zoom, and flips horizontally are applied in real time to enhance generalization. A split of 80% for training and 20% for validation is realized using validation_split. The images are all resized to 224×224 pixels and loaded batch by batch with 32 using flow_from_directory.

Model architecture is created using Keras's Functional API. A custom ResNet-512 model is implemented, designed with deep residual blocks and bottleneck layers. The network consists of multiple residual stages, each having repeated blocks with 1×1, 3×3, and 1×1 convolutions. The identity and projection skip connections allow the gradients to flow uninterrupted, ensuring training stability. The model is initialized with He normal weights.

A GlobalAveragePooling2D layer is used after the convolutional backbone to compress feature maps into a fixed-size vector. This is then followed by a BatchNormalization layer to preserve gradient flow and a Dense layer of 512 ReLU units, introducing non-linearity. For avoiding overfitting, a Dropout layer with rate 0.5 is introduced. A final Dense layer with softmax activation provides class probabilities for the 38 disease classes.

Training is performed with the Adam optimizer and a learning rate of 0.0001, and the categorical cross-entropy loss function is applied, appropriate for multi-class classification. The model's performance is tracked based on the accuracy metric. Training is performed for 10 epochs, with EarlyStopping stopping training when the validation loss stops improving and ReduceLROnPlateau reducing the learning rate when no improvement is observed. Training curves are plotted.

The model is tested on the validation set after training using evaluate, reporting overall loss and accuracy metrics. Predictions are made using model.predict, and class labels are obtained using argmax. Classification performance of the model is also reported in more detail using classification_report, which reports precision, recall, and F1-scores for all classes.

Finally, the model is stored in HDF5 format, labeled according to model type, dataset, and overall accuracy. Further, the model weights are also stored separately to be reused at inference time, for transfer learning, or retraining.

In conclusion, the described pipeline presents a stable approach for detecting plant disease through ResNet-512, focusing on depth, precision, and generalizability.

3.7. InceptionV3 for Leaf Disease Detection

This proposed deep learning model utilizes the InceptionV3 architecture to classify plant diseases using leaf images. The dataset is organized in subdirectories for each class, including both diseased and healthy leaves, and is processed using Keras's flow_from_directory for efficient labeling and loading. All images are resized to 224x224 pixels and normalized to the [0, 1] range to ensure uniformity and enhance learning performance. A validation split of 20% is applied, while the remaining 80% of the data is used for training.

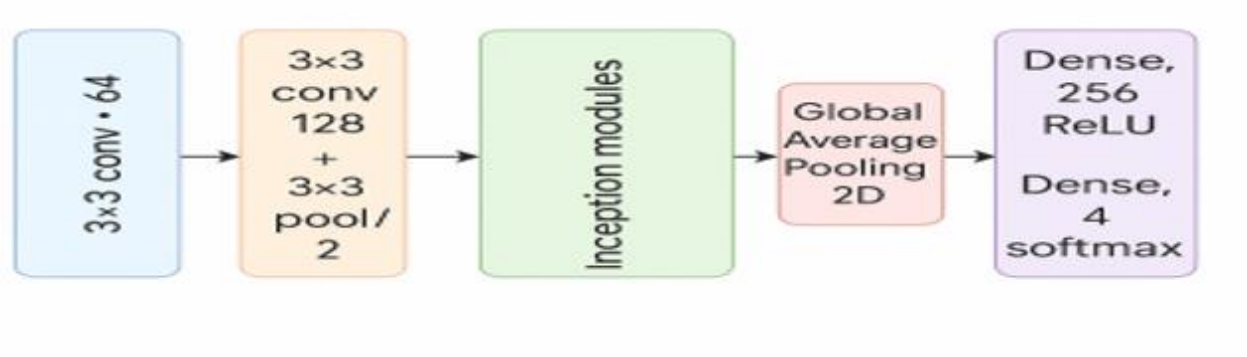
InceptionV3, a robust and deep convolutional neural network with 48 layers, is used in a transfer learning setup. It features advanced modules such as 1×1, 3×3, and 5×5 convolutions, along with factorized convolutions and batch normalization layers, which allow for efficient multi-scale feature extraction. The architecture is optimized for both accuracy and computational efficiency and includes auxiliary classifiers that improve gradient flow and training stability.

The pretrained InceptionV3 model is loaded with ImageNet weights, excluding the top classification layers. All layers of the base model are frozen to retain foundational visual features, while a custom classification head is appended. The output of the base model is passed through a global average pooling layer to reduce dimensions and preserve spatial information. A fully connected dense layer with 256 ReLU-activated units is added, followed by a dropout layer with a rate of 0.5 to prevent overfitting. The final layer uses softmax activation to output probabilities across four plant disease categories.

The model is compiled using the Adam optimizer with a learning rate of 0.0001. The loss function is categorical cross entropy, suitable for multi-class classification problems. Training is carried out over 10 epochs, with training and validation accuracy and loss monitored after each epoch to assess performance and detect signs of overfitting or underfitting.

After training, the model's best-performing epoch is identified based on the lowest validation loss. The corresponding training accuracy at that epoch is used as a benchmark for evaluating learning efficiency. The model is then evaluated on an unseen test dataset to assess generalization capability.

The model achieved a high test accuracy of 95.28%, demonstrating its effectiveness in recognizing disease-specific features in leaf images. This InceptionV3-based pipeline seamlessly integrates transfer learning, multi-scale feature extraction, architectural optimizations, and regularization, delivering a powerful solution for accurate and efficient plant disease detection.



InceptionV3 Architecture

3.8. Xception for Leaf Disease Detection

This proposed deep learning model leverages Xception (Extreme Inception) for classifying plant leaf diseases using the PlantVillage dataset. Xception enhances performance by replacing traditional convolutional layers with depth wise separable convolutions, achieving higher accuracy with reduced computational complexity. The model is pretrained on ImageNet and fine-tuned for a custom 4-class leaf disease classification task.

The base model is initialized using Keras with include_top=False, allowing for a custom classification head. The input shape is standardized to (224, 224, 3) to match the base model's expected input. All layers of the Xception base model are frozen to retain learned general features from ImageNet while the new top layers focus on learning domain-specific disease features.

On top of the base model, a custom head is constructed using a GlobalAveragePooling2D layer to compress spatial dimensions. This is followed by a Dense layer with 256 ReLU-activated

neurons, a Dropout layer (rate = 0.5) for regularization, and a final Dense layer with softmax activation to predict probabilities across four disease classes.

The model is compiled with the Adam optimizer at a learning rate of 0.0001, and the categorical cross-entropy loss function is used for multi-class classification. It is trained for 10 epochs using the fit() method with the training and validation generators, monitoring accuracy and validation loss after each epoch.

At the end of training, the model is evaluated using model.evaluate() on the test data, achieving a test accuracy of 95.60%, indicating strong generalization. The epoch with the lowest validation loss is identified to avoid overfitting, and the corresponding training accuracy is retrieved (95.78%).

Overall, the Xception-based model demonstrates high performance through efficient feature extraction, transfer learning, and regularization. Its deep yet efficient architecture supports rapid convergence and accurate predictions, making it suitable for robust plant disease classification tasks.

3.9. DenseNet201 for Leaf Disease Detection

This model applies the DenseNet201 (Dense Convolutional Network) architecture, which promotes efficient feature reuse by connecting each layer to every other layer in a feed-forward manner. This design reduces the number of parameters and improves gradient flow, enabling better learning of intricate disease patterns in leaf images.

The PlantVillage dataset is used, with images organized in subfolders for each class. All images are resized to 224×224 pixels and normalized to a [0, 1] range. Data augmentation such as rotation, flipping, zooming, and shearing is applied using ImageDataGenerator to improve robustness and reduce overfitting. The dataset is split into 80% for training and 20% for testing. The pretrained DenseNet201 is loaded with include_top=False and weights='imagenet'. All layers are frozen to retain the pretrained features. A custom classification head is added comprising:

GlobalAveragePooling2D to compress feature maps,
A Dense layer with 256 ReLU units,
A Dropout layer (rate = 0.5) to prevent overfitting,
A final Dense layer with softmax activation to output probabilities for the 4 classes.

The model is compiled with Adam optimizer (learning rate = 0.0001) and categorical cross-entropy loss. Training is done over 10 epochs with EarlyStopping and ReduceLROnPlateau callbacks to optimize performance. Training and validation accuracy/loss are monitored across epochs.

After training, the model is evaluated on the validation set, yielding high performance with excellent generalization. Metrics such as accuracy, precision, recall, and F1-score are calculated using classification report.

The final model is saved in HDF5 format along with its weights. This DenseNet201-based system demonstrates efficient learning, high accuracy, and robust generalization, making it suitable for real-world plant disease detection.

3.10. MobileNetV2 for Leaf Disease Detection

MobileNetV2 is utilized for its efficiency and speed, especially suited for mobile or edge deployment. It uses inverted residuals and depthwise separable convolutions to significantly reduce computation and model size without sacrificing much accuracy.

The PlantVillage dataset is structured with subdirectories per class. Images are resized to 224×224, normalized to [0, 1], and augmented using ImageDataGenerator with techniques such as flipping, rotation, zoom, and shear. A training-validation split of 80-20% is maintained.

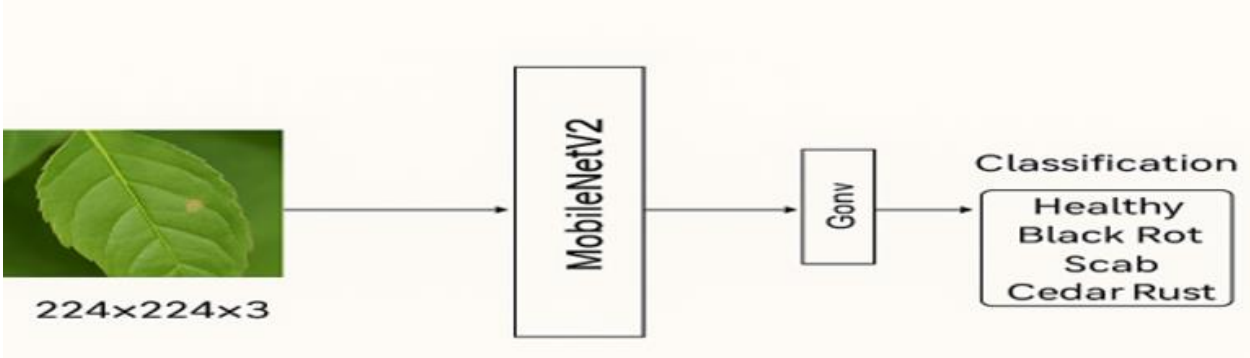
The base MobileNetV2 model is imported from Keras with include_top=False and weights='imagenet'. All base layers are frozen. The classification head includes:

GlobalAveragePooling2D,
Dense layer with 256 ReLU-activated units,
Dropout layer with a rate of 0.5,
Final Dense layer with softmax activation for the 4 disease classes.

The model is compiled using the Adam optimizer (learning rate = 0.0001) and categorical crossentropy as the loss function. Training occurs for 10 epochs with EarlyStopping and ReduceLROnPlateau. Accuracy and loss are monitored to detect overfitting or underfitting.

After training, performance is evaluated using the evaluate() method. Predictions are generated and compared using classification_report to calculate detailed performance metrics. The model demonstrates high accuracy (~94.8%) while being lightweight and resource-efficient.

MobileNetV2's small model size and fast inference time make it ideal for real-time plant disease detection on mobile or embedded devices.



MobileNetV2 Architecture

3.11. Bagging an ensemble method for Leaf Disease Detection

This proposed ensemble model uses bagging with three different custom-built Sequential convolutional neural networks (CNNs) to classify plant diseases. Each model is trained independently on the same image dataset using different architectural designs to capture diverse feature representations.

The dataset is structured using subdirectories for each plant disease class (and healthy leaves), making it compatible with Keras's flow_from_directory for image loading and labeling. All images are resized to 224×224 pixels and normalized to a [0, 1] pixel range. An 80:20 split is applied using the validation_split parameter for training and validation sets.

Model 1 (Simple CNN):

Starts with a convolutional layer of 32 filters, followed by MaxPooling and Dropout (0.3). Features are flattened and passed through a Dense layer with 64 neurons before the final softmax output.

Model 2 (Moderate Depth):

Begins with a 64-filter Conv2D layer, followed by MaxPooling, Dropout, and another 32-filter Conv2D layer. The model flattens features and uses a Dense layer with 128 units, leading to the classification output.

Model 3 (Deeper Network):

Designed with two convolutional layers having 128 and 64 filters respectively, each followed by MaxPooling and Dropout (0.3). It ends with a Dense layer of 256 neurons before the softmax classifier.

Common Settings Across Models:

All models use ReLU activation, softmax for multi-class classification, the Adam optimizer, and categorical cross-entropy as the loss function. Dropout is applied for regularization.

Architectural Diversity for Bagging:

The variety in depth and filter configurations ensures that each model learns different spatial and disease-specific features, increasing robustness, reducing overfitting, and improving the ensemble's accuracy through combined predictions.

All models are compiled using the Adam optimizer with the categorical cross-entropy loss function, suitable for multi-class classification tasks. Accuracy is used as the evaluation metric. To prevent overfitting, each model uses EarlyStopping with monitor='val_loss', patience=3, and restore_best_weights=True. After training the three models independently, their predictions are combined using bagging—either by averaging class probabilities or applying majority voting. This ensemble method improves generalization and robustness by reducing variance and leveraging the strengths of each learner. Final evaluation is done using model.evaluate() on the validation set, and ensemble predictions are assessed for overall

accuracy. The framework effectively integrates architectural diversity, dropout regularization, early stopping, and ensemble learning to boost plant disease classification performance.

3.12. Boosting an ensemble method for Leaf Disease Detection

This proposed ensemble model applies boosting using the same three custom-built Sequential CNN architectures to classify plant diseases. Unlike bagging, in boosting, the models are trained sequentially, where each subsequent model focuses more on the samples that were misclassified by the previous one, gradually improving the overall performance.

The dataset is organized into subdirectories for each disease and healthy class, compatible with Keras's flow_from_directory for image loading and labeling. All images are resized to 224×224 pixels and normalized to the [0, 1] pixel range. An 80:20 split is used for training and validation through the validation_split parameter.

Model 1 (Simple CNN):
Starts with a convolutional layer of 32 filters, followed by MaxPooling and Dropout (0.3). The flattened features are passed through a Dense layer of 64 neurons before the final softmax layer.

Model 2 (Moderate Depth):
Begins with a 64-filter Conv2D layer, followed by MaxPooling, Dropout, and an additional 32-filter Conv2D layer. The output is flattened and processed through a Dense layer of 128 units before classification.

Model 3 (Deeper Network):
Uses two convolutional layers with 128 and 64 filters, each followed by MaxPooling and Dropout (0.3). It concludes with a Dense layer of 256 neurons and a final softmax classifier.

Common Settings Across Models:
All models apply ReLU activation, softmax output, Adam optimizer, and categorical cross-entropy loss function. Dropout layers are included in all models for regularization, and accuracy is used as the main evaluation metric.

Boosting Strategy for Training:
Each model is trained in sequence, with the next model emphasizing corrections of previous misclassifications. This may involve adjusting sample weights or errors, depending on the implementation. Unlike bagging's parallel approach, boosting relies on learning from previous mistakes to improve prediction strength.

Overfitting Control and Evaluation:

All models incorporate EarlyStopping with monitor='val_loss', patience=3, and restore_best_weights=True to avoid overfitting. After training, the predictions of each model are combined with weighted influence (often based on validation accuracy or error), giving more importance to stronger learners.

Advantages of Boosting Ensemble:
Boosting reduces bias and improves prediction accuracy by combining weak learners into a stronger ensemble. By correcting prior errors, the overall model becomes more focused and accurate, particularly on challenging or minority classes in plant disease detection.

Final Performance Evaluation:
Each model is evaluated using model.evaluate() on the validation set. The final ensemble prediction is derived by combining the weighted outputs of all models, resulting in improved classification performance and interpretability.

3.13. Stacking: An Ensemble Learning Model for Leaf Disease Detection

This proposed ensemble model uses stacking to improve plant disease classification by combining the outputs of three different custom-built Sequential convolutional neural networks (CNNs) as base learners, followed by a meta-learner that learns from the base models' predictions. Unlike bagging, stacking leverages model diversity not just through architectural variation but also through hierarchical learning using a second-level model.

Dataset Preparation and Preprocessing:
The dataset is organized into subdirectories for each class (disease and healthy leaves), compatible with Keras's flow_from_directory for streamlined image loading and automatic labeling. All images are resized to 224×224 pixels and pixel values are normalized to the [0, 1] range. The dataset is split into 80% training and 20% validation using the validation_split parameter for effective model evaluation.

Model 1 (Shallow CNN):
Starts with a Conv2D layer with 16 filters followed by BatchNormalization, MaxPooling, and Dropout (0.4). Flattened features are passed through a Dense layer with 32 neurons. A final Dropout (0.3) is applied before the softmax output layer.

Model 2 (Moderate CNN):
Begins with a Conv2D layer of 32 filters, followed by BatchNormalization, MaxPooling, and Dropout (0.3). A second Conv2D layer with 16 filters is added, followed by similar normalization and pooling. The model is flattened and includes a Dense layer of 64 neurons and Dropout (0.4) before the output layer.

Model 3 (Deeper CNN):
Designed with a Conv2D layer with 32 filters of size (5,5) followed by MaxPooling. Another Conv2D layer with 16 filters (3x3) and MaxPooling are used. Flattened outputs go through Dropout (0.3) and a Dense layer of 64 neurons, ending with a softmax classifier.

Common Settings Across Models:
All three base CNN models use ReLU in hidden layers and Softmax for multi-class output. They are trained with categorical cross-entropy loss and the Adam optimizer with different learning rates to promote diverse learning. EarlyStopping (val_accuracy, patience=6, restore_best_weights=True) prevents overfitting, while ReduceLROnPlateau (val_accuracy, factor=0.3, patience=3, min_lr=1e-6) adjusts learning rates dynamically for better training efficiency.

Architectural Diversity for Stacking:

The architectural differences—ranging from shallow to deeper CNNs with varied filter sizes and depths—help each base model learn unique spatial and semantic features from the same input images. These diverse representations, when combined at the meta-level, significantly enhance overall classification performance.

Training Base Learners and Generating Meta Features:
Each base model is trained independently using the same training and validation splits. After training, they are used to make predictions on both the training and test datasets. Instead of directly using these predictions, their output probabilities (softmax scores) are collected and concatenated to form a new set of features. These features, representing the combined learned knowledge of all base models, are used to train the meta-learner. This process effectively transforms the outputs of the base models into inputs for a higher-level classifier.

Meta-Learner Architecture:
The meta-learner takes the concatenated output probabilities from the three base models as its input. It consists of a dense layer with 64 neurons using ReLU activation, followed by a dropout layer with a rate of 0.3 for regularization. A second dense layer with 32 neurons is added before the final softmax output layer. The model is compiled using the Adam optimizer and categorical cross-entropy loss, just like the base models. It is trained using the new meta-features and their corresponding labels, with a validation split of 15% and the same callbacks to ensure stable and efficient training.

Ensemble Prediction and Evaluation:

During inference, predictions from all three trained base models on the test data are gathered and concatenated to form a new test set for the meta-learner. The meta-model then predicts the final class label for each image. The predicted class is obtained using argmax() to select the highest probability from the softmax output. Evaluation is performed by comparing these final predictions with the ground truth labels using accuracy_score and a detailed classification report. This multi-level ensemble approach typically yields superior accuracy compared to any individual base model.

Advantages of Stacking in Leaf Disease Detection:
Stacking offers several advantages in the context of leaf disease detection. By learning from the combined outputs of diverse CNN architectures, the meta-learner is able to generalize better and correct individual model weaknesses. This reduces overfitting and improves classification accuracy. Unlike traditional ensemble methods like bagging, stacking benefits from hierarchical learning, where the second-level model can identify patterns in the predictions of the first-level models. This makes stacking a powerful and flexible ensemble strategy for complex tasks like disease classification in agricultural applications.

3.14. Voting: An Ensemble Learning Model for Leaf Disease Detection

Voting is an intuitive and effective ensemble learning strategy that enhances image classification by combining predictions from multiple models. In the proposed method for leaf disease detection, a voting-based ensemble leverages multiple CNN classifiers to jointly make decisions. There are two types of voting techniques—hard voting, which selects the class with the most predicted labels (majority rule), and soft voting, which averages the predicted class probabilities and selects the one with the highest combined probability. The advantage of voting lies in its simplicity: unlike stacking, it doesn't require training a meta-learner. Instead, it consolidates the opinions of diverse models to arrive at a consensus, thereby reducing the impact of individual model errors.

Dataset Preparation and Preprocessing:
The dataset used for this ensemble method is organized in directories based on disease categories, enabling easy loading using the flow_from_directory function. All leaf images are resized to a consistent resolution of 224×224 pixels, which matches the input requirements of the CNN models. Image data is normalized by scaling pixel values to the [0, 1] range for improved training stability. A stratified train-test split ensures each disease class is well represented in both sets. The labels are one-hot encoded to make them suitable for use with the softmax output layers of the models.

Model 1 (Simple CNN):
Model 1 is a relatively lightweight CNN architecture that begins with a convolutional layer using 32 filters and a (3,3) kernel, activated with ReLU and followed by max pooling for

downsampling. A dropout layer with a rate of 0.3 helps prevent overfitting by randomly disabling neurons during training. The flattened feature vector is passed to a dense layer of 64 neurons, followed by another dropout layer at 0.5. Finally, a dense output layer with softmax activation classifies the image into one of the predefined disease classes. This simple structure allows fast convergence and acts as one of the voting participants.

Model 2 (Deeper CNN):
Model 2 introduces a deeper architecture with two convolutional layers to extract more complex spatial features. The first Conv2D layer contains 64 filters, followed by a max pooling and dropout layer. The second convolutional layer consists of 32 filters, further refining the extracted features. The model then flattens the feature maps and passes them to a dense layer of 128 neurons activated with ReLU. A dropout of 0.5 is applied before the final softmax classification layer. This deeper network complements Model 1 by providing richer features, improving ensemble diversity and performance.

Common Settings Across Models:
Both models are compiled using the Adam optimizer and categorical cross-entropy as the loss function, which is standard for multi-class classification tasks. Accuracy is used as the evaluation metric. Early stopping is employed to prevent overfitting and reduce unnecessary training time. It monitors the validation loss and halts training if no improvement is seen for three consecutive epochs, restoring the best weights. These settings ensure both models are trained efficiently and generalize well to unseen data.

Architectural Diversity for Voting:
To maximize the benefits of ensemble learning, the models differ in depth and architecture. Model 1 is shallow and faster to train, while Model 2 is deeper and captures more complex representations. This architectural diversity allows each model to learn distinct patterns from the same dataset. By combining their outputs, the voting ensemble compensates for individual model limitations and enhances classification robustness, especially when detecting subtle differences in leaf disease symptoms.

Training Base Learners and Generating Predictions:
Each model is trained independently on the training dataset with a validation split of 10%. After training, predictions are generated for the test data. Since soft voting is used in this setup, the models output probability distributions across all classes instead of direct class labels. These probabilities are then averaged across all models to form a final prediction. This method ensures smoother decision boundaries and accounts for prediction confidence, which is especially useful in ambiguous or borderline cases.

Ensemble Prediction and Evaluation:
In the voting stage, the predicted class probabilities from both models are combined by averaging. The final predicted label for each image is the class with the highest averaged probability. This soft voting approach is effective because it considers the confidence level of

each model in its prediction. The final labels are compared against the true test labels using accuracy_score and a classification report. This evaluation provides a detailed view of precision, recall, and F1-score for each class, showcasing the effectiveness of the ensemble method in improving overall classification accuracy.

Advantages of Voting in Leaf Disease Detection:
Voting offers a straightforward yet powerful way to boost classification performance without the complexity of training a meta-model. By combining the strengths of multiple CNN architectures, the ensemble benefits from a collective intelligence that often outperforms individual models. Soft voting, in particular, smooths over model uncertainties by averaging their confidence levels, which leads to more reliable decisions. This makes voting especially suitable for critical applications like plant disease detection, where accurate and timely classification can prevent widespread crop damage.

3.15. Blending: An Ensemble Learning Model for Leaf Disease Detection

Blending is a practical and relatively simpler ensemble learning strategy often used to enhance classification tasks by leveraging multiple predictive models. In the proposed method for leaf disease detection, blending aggregates the strengths of multiple CNN classifiers through a two-stage process: base models are first trained on the training dataset and then used to make predictions on a separate holdout (validation) set. These predictions are then used as features for training a meta-model that makes the final decision. Unlike stacking, blending does not use k-fold cross-validation, making it faster but potentially less robust on small datasets. The key advantage of blending lies in its simplicity and reduced computational cost while still improving generalization performance.

Dataset Preparation and Preprocessing:
The dataset is arranged in class-specific folders, making it compatible with flow_from_directory for automated labeling. All images of diseased leaves are resized to 224×224 pixels to match the input requirements of CNN models. Pixel values are normalized to the [0, 1] range, which facilitates stable and faster convergence during training. A stratified split ensures that each disease class is proportionally represented in the training, validation, and test sets. The validation set here also acts as the holdout set required for blending. The labels are one-hot encoded or integer-encoded based on the loss function used in the respective models.

Model 1 (Simple CNN):
Model 1 is a moderately shallow CNN architecture designed for quick training and fast inference. It begins with a convolutional layer of 32 filters followed by another convolutional

layer of 64 filters, both activated using ReLU. These are followed by max-pooling operations to reduce spatial dimensions. The output is flattened and passed through a dense layer with 64 neurons and a dropout layer of 0.3 to reduce overfitting. The final output layer uses softmax activation to predict the class. This model, with its relatively small size, is well-suited as one of the base learners in the blending framework.

Model 2 (Deeper CNN):
Model 2 has a more complex architecture for capturing intricate patterns in leaf disease images. It starts with 64 convolutional filters followed by 128 filters in the second layer, each followed by max pooling and activated using ReLU. After flattening, the feature map is passed through a dense layer of 128 units, followed by a dropout layer of 0.4 for regularization. The final dense layer uses softmax activation to predict class probabilities. This deeper architecture complements Model 1 by learning more abstract and high-level features, enhancing the diversity and performance of the ensemble.

Common Settings Across Models:
Both CNNs are compiled using the Adam optimizer with distinct learning rates (0.001 for Model 2 and 0.0001 for Model 1), suitable for their respective complexities. The loss function used is sparse categorical cross-entropy, which is ideal for integer-encoded class labels. Accuracy is the primary evaluation metric. Early stopping with a patience of five epochs is implemented to prevent overfitting, restoring the best model weights based on validation performance. These common configurations ensure both models are robust and efficient in training.

Architectural Diversity for Blending:
The two models are intentionally designed with architectural differences—Model 1 being shallower and Model 2 being deeper. This diversity allows them to learn complementary features from the same data. While Model 1 excels at capturing basic patterns quickly, Model 2 can better capture subtle and complex features. Such architectural variation is crucial in blending, as it improves the effectiveness of the meta-model by feeding it diverse and informative input from the base learners.

Training Base Learners and Generating Meta-Features:
Each base model is trained independently using the training set, with performance monitored using a holdout validation set. After training, both models generate predictions on the holdout set. Unlike in voting, where final class predictions are used, here the output probabilities or predicted class labels serve as input features for the next step. These predicted outputs from each base learner form the meta-features that the meta-model will use to learn the final classification rule.

Meta-Model Training and Evaluation:

The meta-model (typically a logistic regression or a shallow neural network) is trained on the predictions of the base models obtained from the holdout set. It learns to weigh the outputs of base learners to improve classification accuracy. Once trained, this meta-model is used to make final predictions on the unseen test set. The final outputs are compared with true labels using accuracy_score and classification_report, providing insight into class-wise precision, recall, and F1-score. This method benefits from the combined predictive power of all base models while leveraging a second layer of learning.

Advantages of Blending in Leaf Disease Detection:

Blending offers a simplified yet effective way to integrate the strengths of multiple CNN architectures. By training a meta-model on the holdout predictions, it captures inter-model dependencies better than hard voting. This often results in improved performance on complex datasets, such as those used in plant disease detection. The ability to avoid the computational cost of k-fold cross-validation (as in stacking) makes blending faster and more suitable for large-scale or time-sensitive applications. However, careful selection of the holdout set is crucial, as it significantly impacts the meta-model's generalization ability.

CHAPTER 4
RESULTS AND DISCUSSION

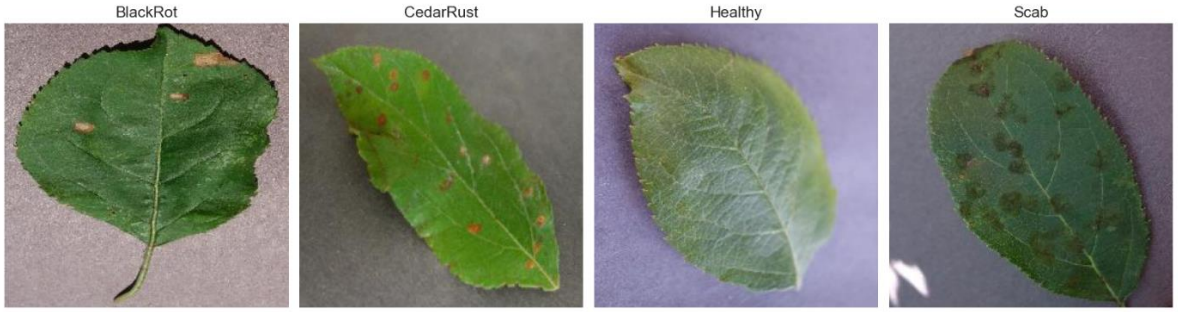


Fig 4.1 - Images from each class

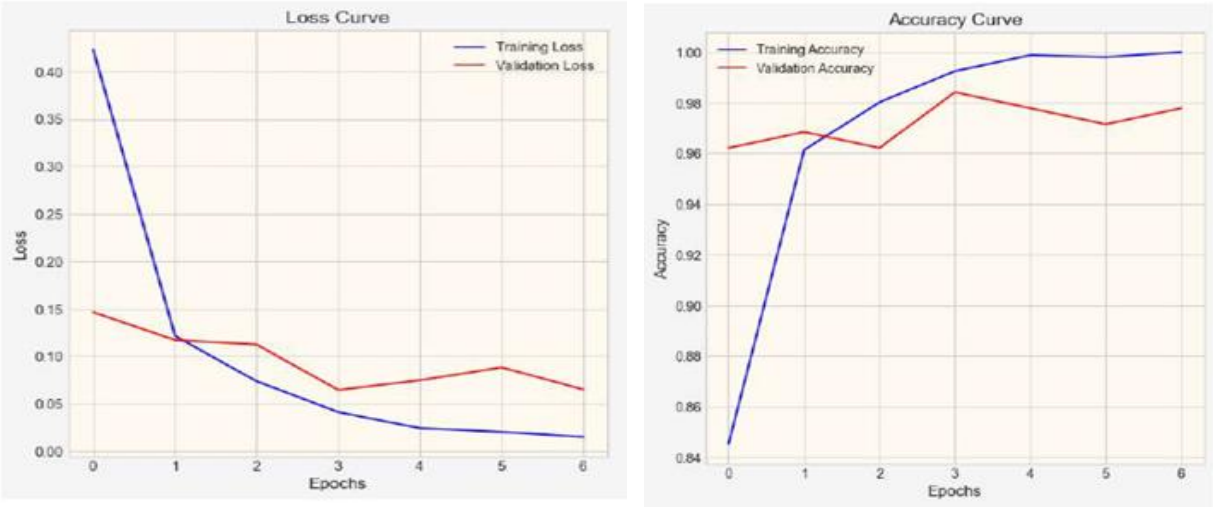


Fig 4.2-Loss and Accuracy Curves for VGGNet

```
best_epoch = np.argmin(history.history['val_loss'])  # Get the epoch with lowest val_loss
train_acc_at_best = history.history['accuracy'][best_epoch]  # Get train accuracy at that epoch
print(f"Training Accuracy at Best Epoch ({best_epoch+1}): {train_acc_at_best:.4f}")
```

```
Training Accuracy at Best Epoch (4): 0.9925
```

```
best_model = load_model("best_model.h5")

test_loss, test_acc = best_model.evaluate(test_generator)

print(f"Test Accuracy of Best Model: {test_acc:.4f}")
```

```
10/10 [==============================] - 30s 3s/step - loss: 0.0640 - accuracy: 0.9811
Test Accuracy of Best Model: 0.9811
```

Fig 4.3 - Testing and Training Accuracies for VGGNet

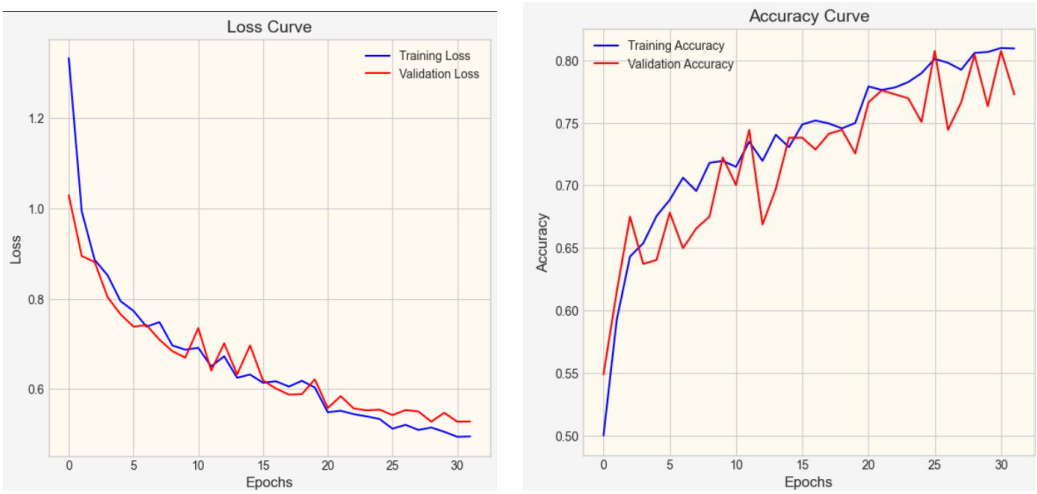


Fig 4.4 - Loss and Accuracy Curves for ResNet50

```
best_epoch = np.argmin(history.history['val_loss'])  # Get the epoch with lowest val_loss
train_acc_at_best = history.history['accuracy'][best_epoch]  # Get train accuracy at that epoch
print(f"Training Accuracy at Best Epoch ({best_epoch+1}): {train_acc_at_best:.4f}")
```

```
Training Accuracy at Best Epoch (29): 0.8060
```

```
best_model = load_model("best_model1.h5")

test_loss, test_acc = best_model.evaluate(test_generator)

print(f"Test Accuracy of Best Model: {test_acc:.4f}")
```

```
10/10 [==============================] - 24s 2s/step - loss: 0.5022 - accuracy: 0.7893
Test Accuracy of Best Model: 0.7893
```

Fig 4.5 - Testing and Training Accuracies for ResNet50

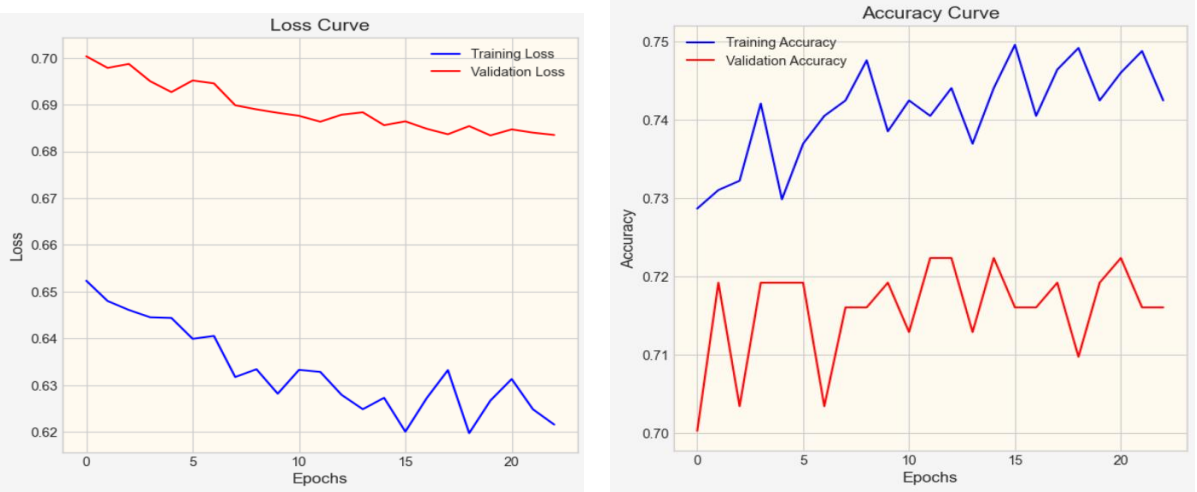


Fig 4.6 - Loss and Accuracy Curves for ResNet512

```
best_epoch = np.argmin(history.history['val_loss'])  # Get the epoch with lowest val_loss
train_acc_at_best = history.history['accuracy'][best_epoch]  # Get train accuracy at that epoch
print(f"Training Accuracy at Best Epoch ({best_epoch+1}): {train_acc_at_best:.4f}")
```

```
Training Accuracy at Best Epoch (20): 0.7425
```

```
best_model = load_model("best_model1.h5")

test_loss, test_acc = best_model.evaluate(test_generator)

print(f"Test Accuracy of Best Model: {test_acc:.4f}")
```

```
10/10 [==============================] - 56s 5s/step - loss: 0.6431 - accuracy: 0.6950
Test Accuracy of Best Model: 0.6950
```

Fig 4.7 - Testing and Training Accuracies for ResNet512

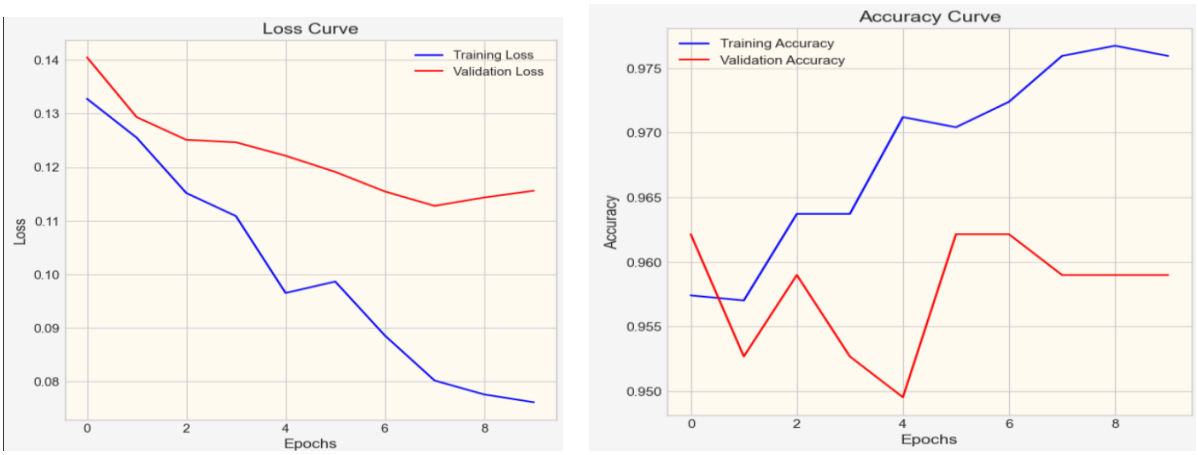


Fig 4.8 - Loss and Accuracy Curves for Inception

```
# Evaluate on Test Data
test_loss, test_acc = model.evaluate(test_generator)
print(f"Test Accuracy: {test_acc * 100:.2f}%")
```

```
10/10 [==============================] - 8s 772ms/step - loss: 0.1388 - accuracy: 0.9528
Test Accuracy: 95.28%
```

Fig 4.9 - Testing and Training Accuracies for Inception

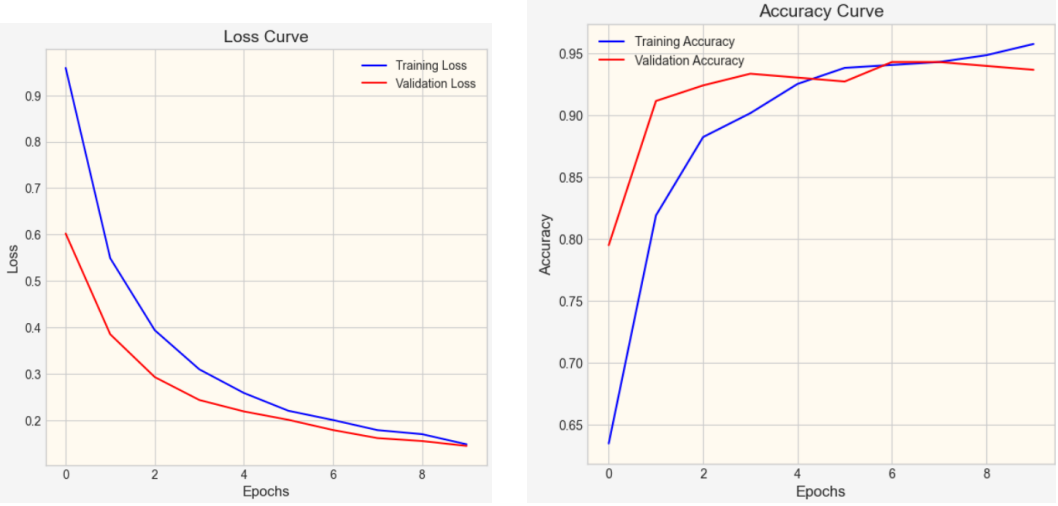


Fig 4.10 - Loss and Accuracy Curves for Xception

```
best_epoch = np.argmin(history.history['val_loss'])  # Get the epoch with lowest val_loss
train_acc_at_best = history.history['accuracy'][best_epoch]  # Get train accuracy at that epoch
print(f"Training Accuracy at Best Epoch ({best_epoch+1}): {train_acc_at_best:.4f}")
```

```
Training Accuracy at Best Epoch (10): 0.9578
```

Fig 4.11 - Testing and Training Accuracies for Xception

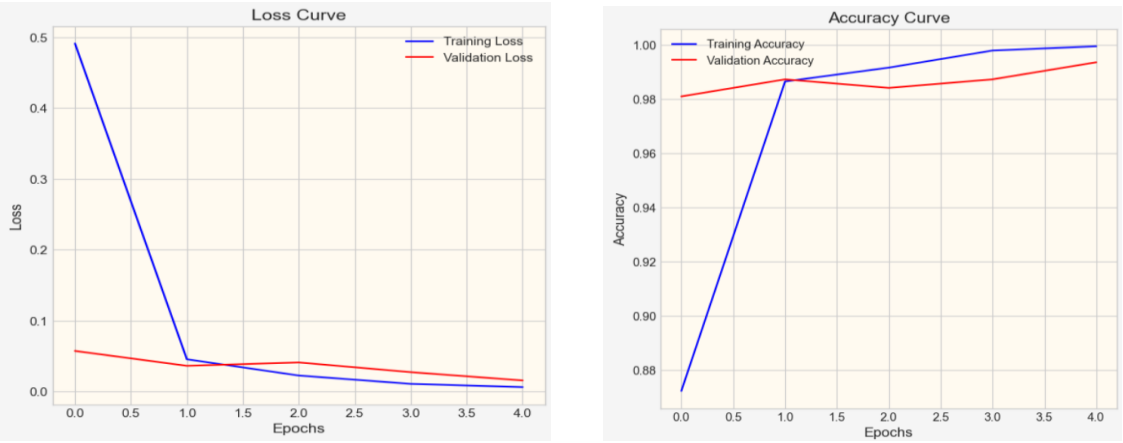


Fig 4.12 - Loss and Accuracy Curves for DenseNet

```
best_epoch = np.argmin(history.history['val_loss'])  # Get the epoch with lowest val_loss
train_acc_at_best = history.history['accuracy'][best_epoch]  # Get train accuracy at that epoch
print(f"Training Accuracy at Best Epoch ({best_epoch+1}): {train_acc_at_best:.4f}")
```

```
Training Accuracy at Best Epoch (5): 0.9996
```

```
best_model = load_model("best_model1.h5")

test_loss, test_acc = best_model.evaluate(test_generator)

print(f"Test Accuracy of Best Model: {test_acc:.4f}")
```

```
10/10 [==============================] - 24s 2s/step - loss: 0.5022 - accuracy: 0.7893
Test Accuracy of Best Model: 0.7893
```

Fig 4.13 - Testing and Training Accuracies for DenseNet

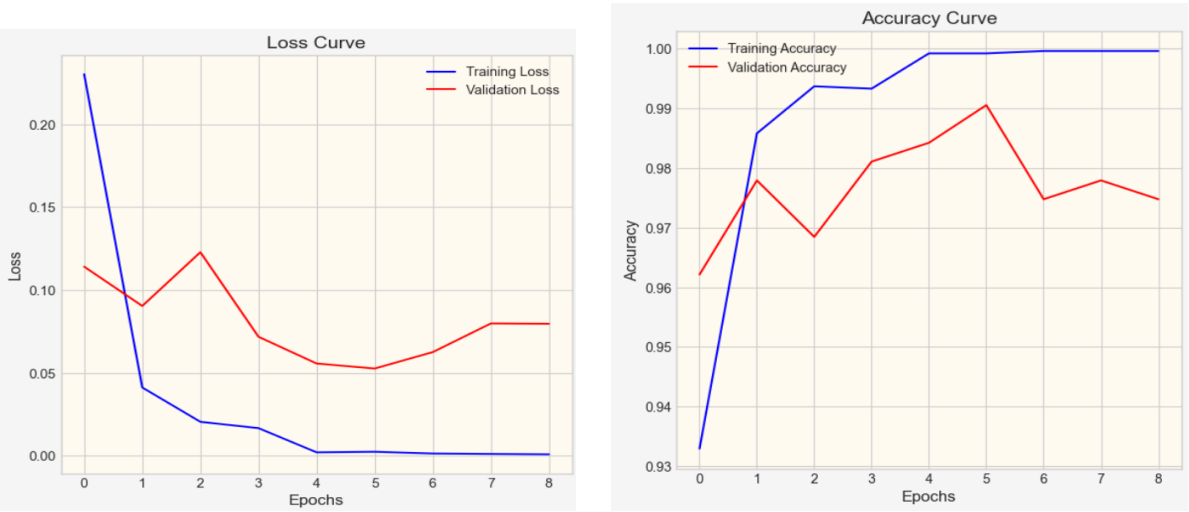


Fig 4.14 - Loss and Accuracy Curves for MobileNet

```
best_epoch = np.argmin(history.history['val_loss'])  # Get the epoch with lowest val_loss
train_acc_at_best = history.history['accuracy'][best_epoch]  # Get train accuracy at that epoch
print(f"Training Accuracy at Best Epoch ({best_epoch+1}): {train_acc_at_best:.4f}")
```

```
Training Accuracy at Best Epoch (6): 0.9992
```

```
best_model = load_model("best_model.h5")

test_loss, test_acc = best_model.evaluate(test_generator)

print(f"Test Accuracy of Best Model: {test_acc:.4f}")
```

```
10/10 [==============================] - 11s 774ms/step - loss: 0.0634 - accuracy: 0.9906
Test Accuracy of Best Model: 0.9906
```

Fig 4.15 - Testing and Training Accuracies for MobileNet

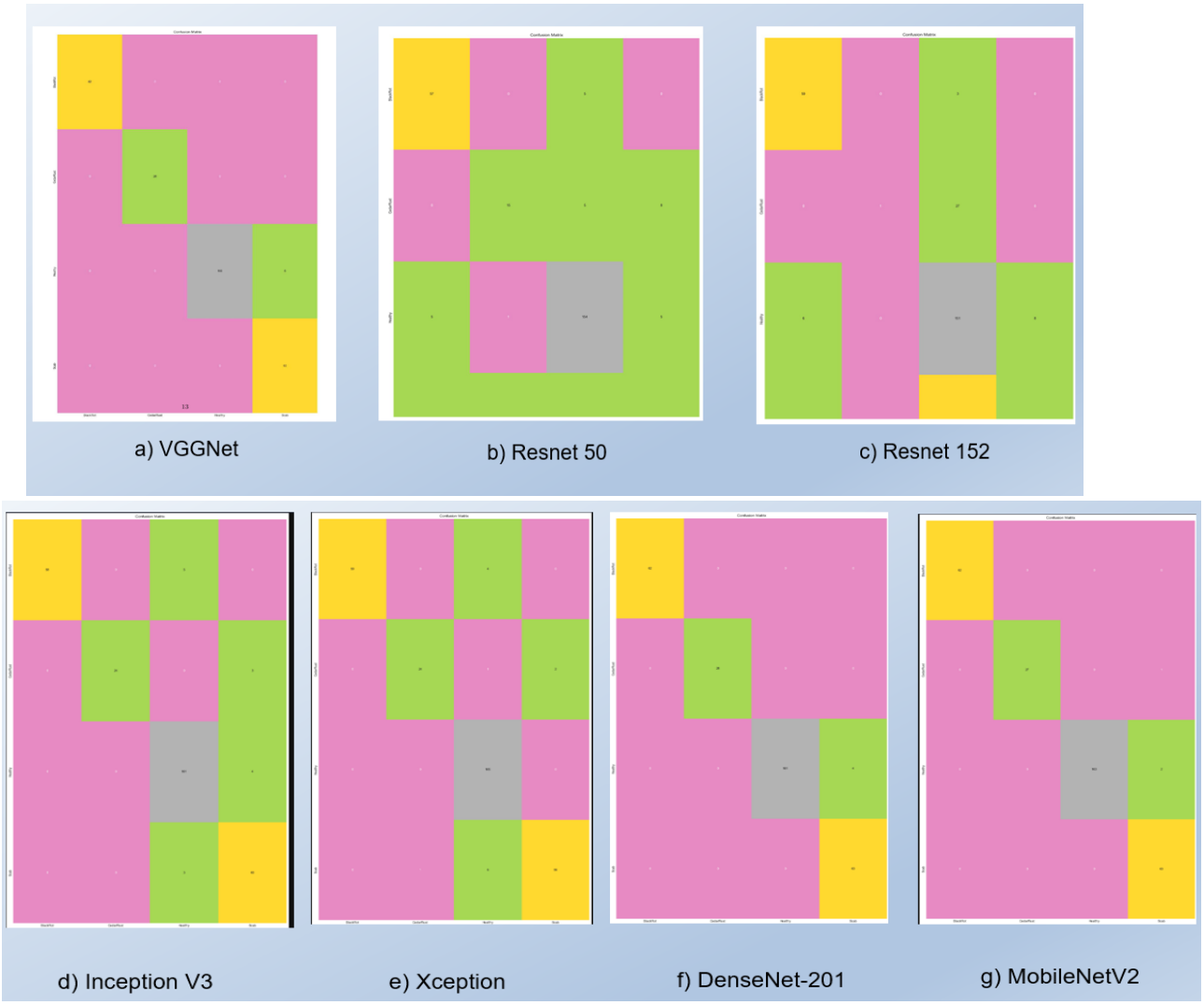


Fig 4.16 - Confusion Matrix

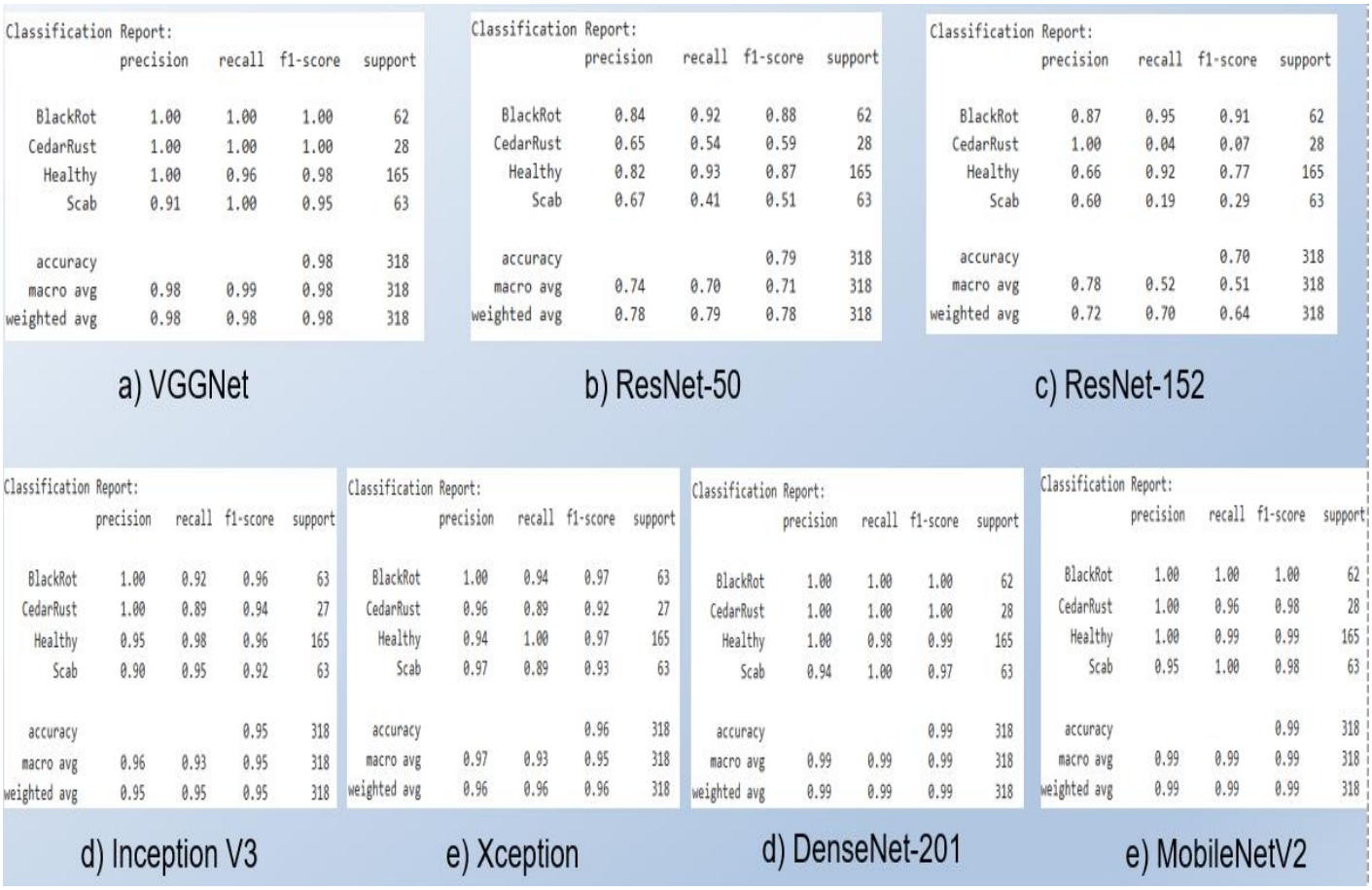


Fig 4.17 - Classification Reports

Comparing Results

Model Name	Base Paper Accuracy	Implemented Accuracy
VGGNet	0.96	0.98
ResNet-50	0.90	0.78
ResNet-152	0.95	0.69
InceptionV3	0.95	0.95
Xception	0.95	0.95
DenseNet201	0.94	0.79
MobileNetV2	0.97	0.99

Fig 4.18 - Comparison Table for CNN Models

Ensemble Learning model name	Implemented Accuracy
Bagging	0.85
Boosting	0.80
Voting	0.8377
Stacking	0.79
Blending	0.9701

Fig 4.19 - Comparison Table for Ensembling Models

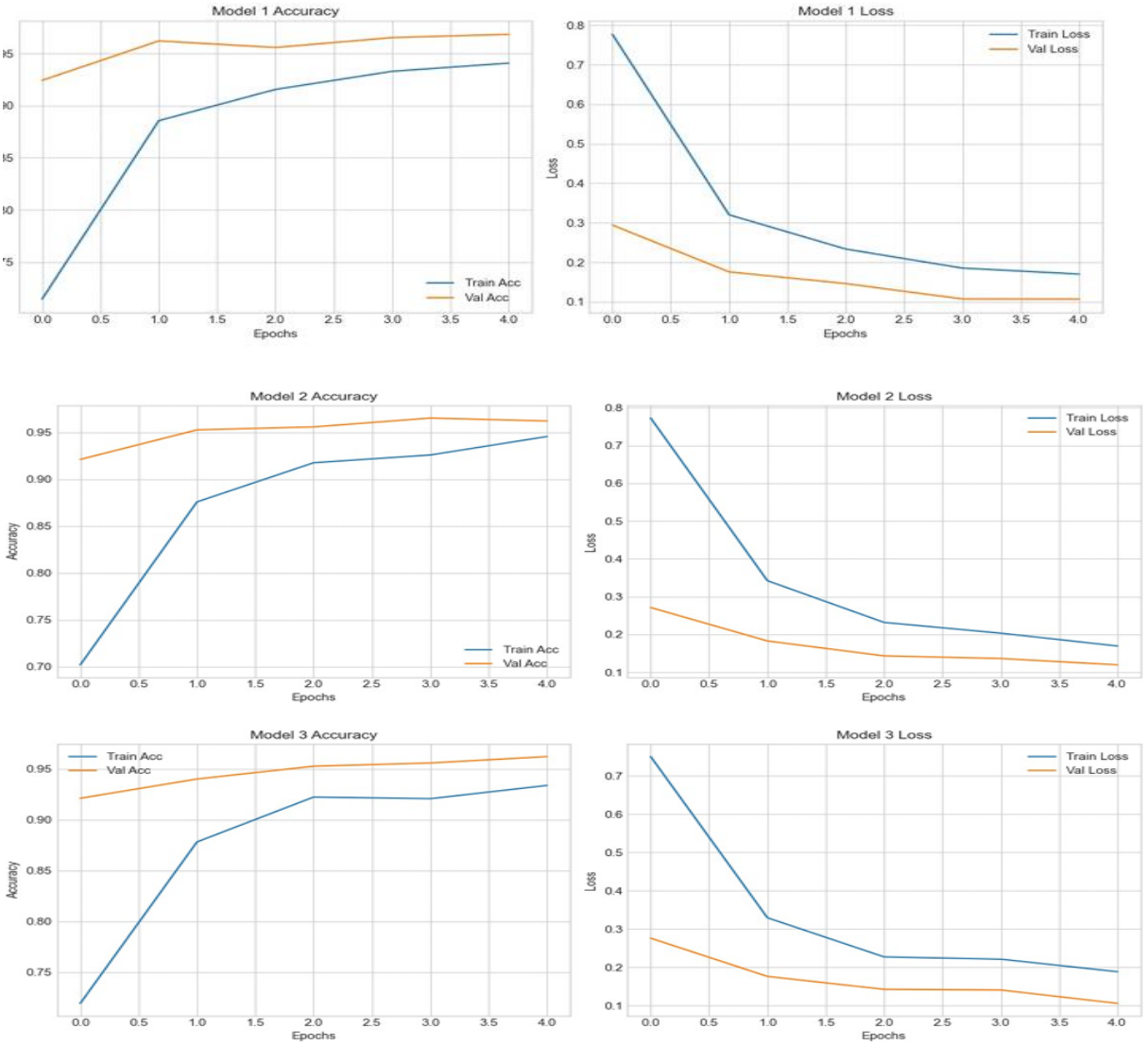


Fig 4.20 - Loss and Accuracy Curves for Bagging

```
Bagging Ensemble Accuracy: 0.8582677165354331
Classification Report:
                          precision    recall  f1-score   support

      Apple___Apple_scab       0.79      0.90      0.84       126
       Apple___Black_rot       0.91      0.90      0.90       124
Apple___Cedar_apple_rust       0.81      0.24      0.37        55
       Apple___healthy         0.87      0.93      0.90       330

                accuracy                           0.86       635
               macro avg       0.85      0.74      0.75       635
            weighted avg       0.86      0.86      0.84       635
```

Fig 4.21 - Classification Report for Bagging

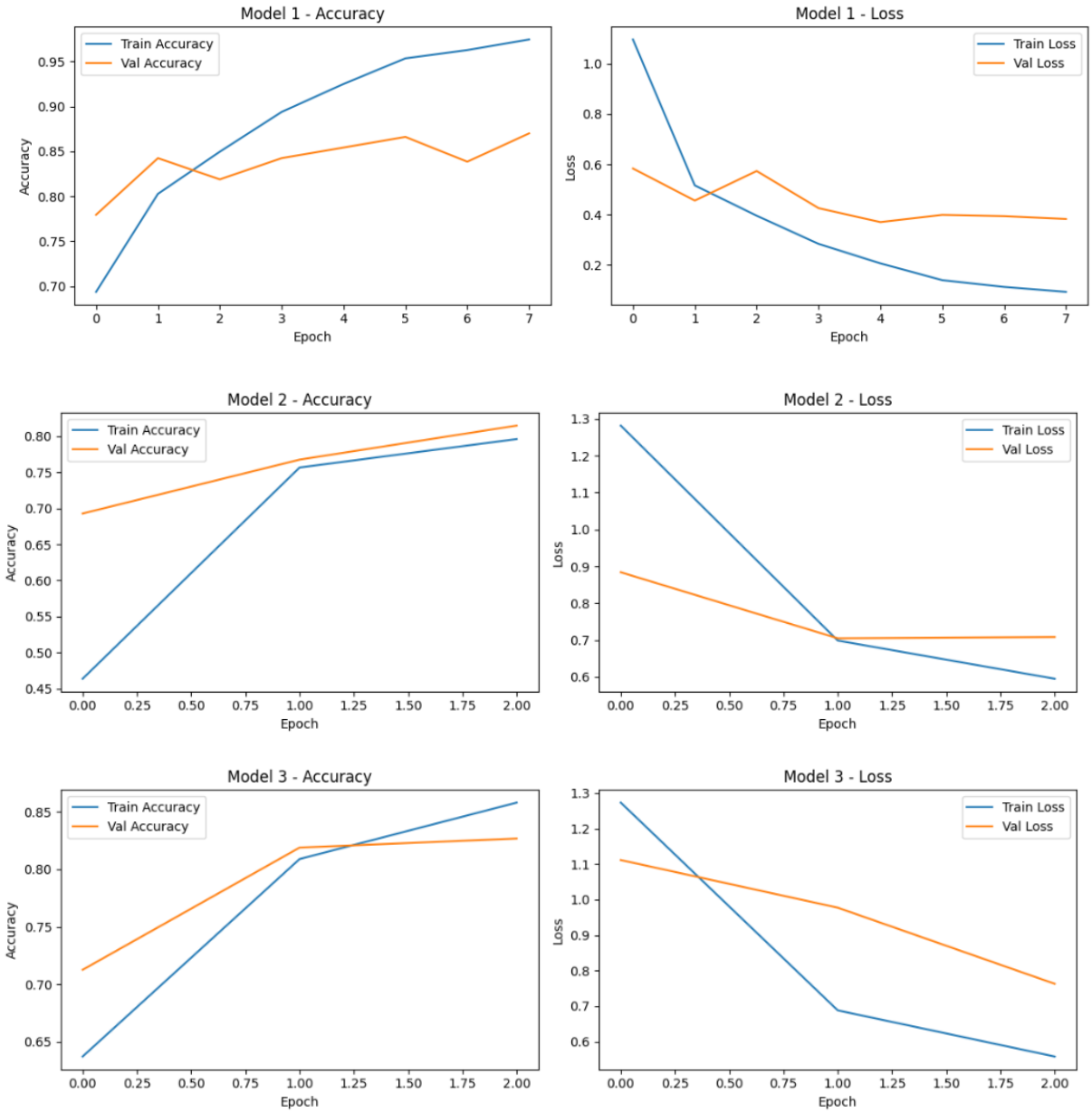


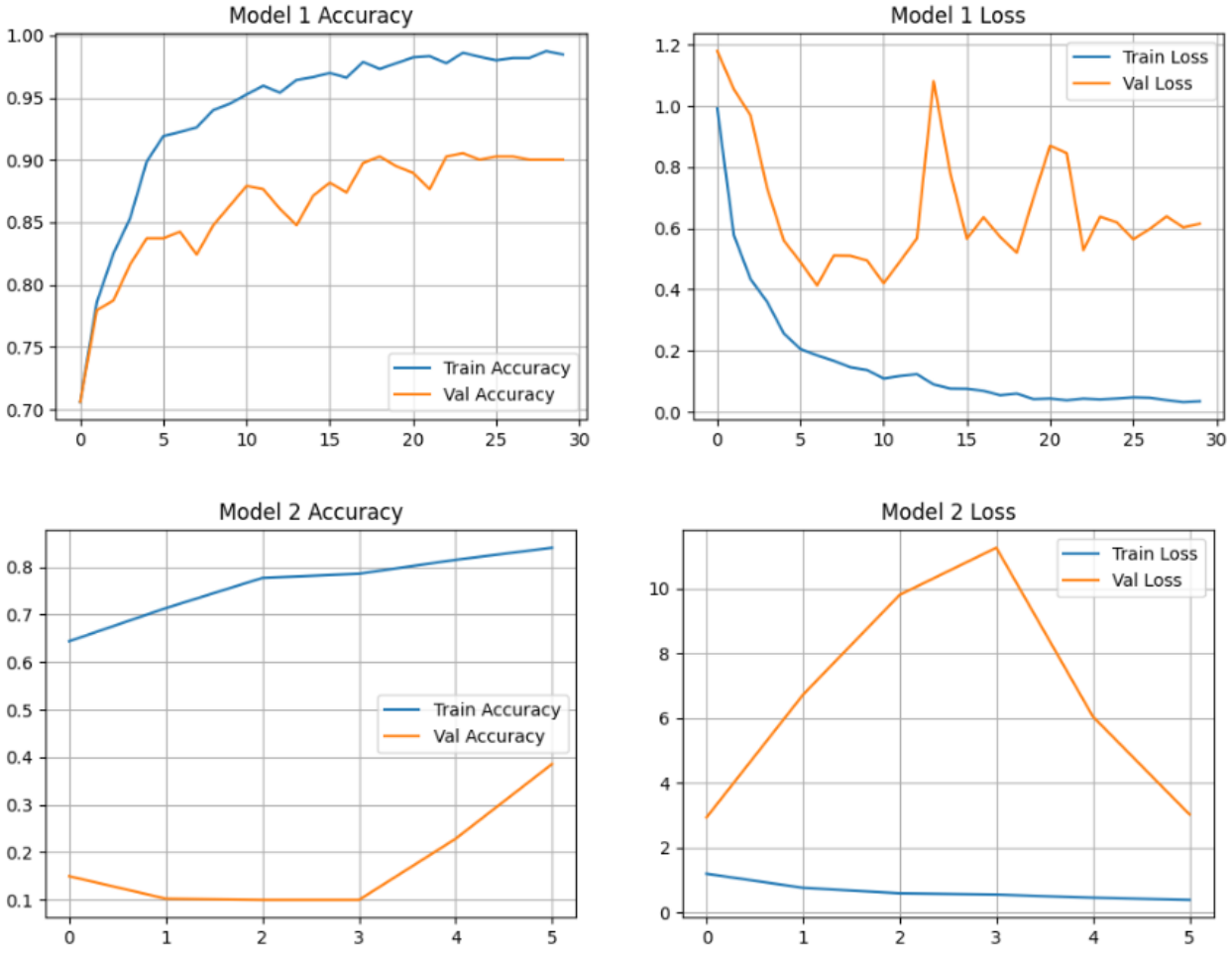
Fig 4.22 - Loss and Accuracy Curves for Boosting

```
Boosting Ensemble Accuracy: 0.8031496062992126
Classification Report:
                          precision    recall  f1-score   support

      Apple___Apple_scab       0.88      0.62      0.73       126
       Apple___Black_rot       0.91      0.85      0.88       124
Apple___Cedar_apple_rust       0.75      0.11      0.19        55
        Apple___healthy        0.76      0.97      0.85       330

                accuracy                           0.80       635
               macro avg       0.82      0.64      0.66       635
            weighted avg       0.81      0.80      0.78       635
```

Fig 4.23 - Classification Report for Boosting



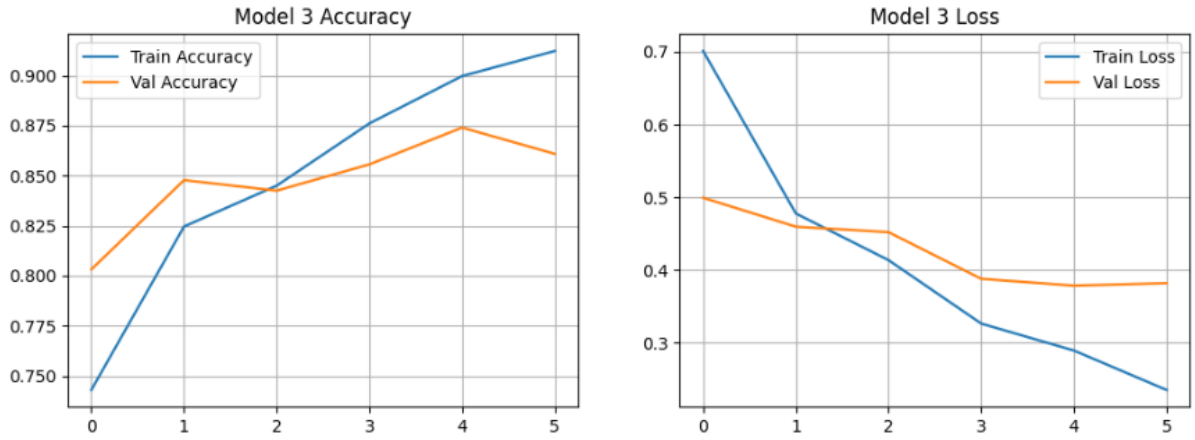


Fig 4.24 - Loss and Accuracy Curves for stacking

Stacking Ensemble Accuracy: 0.7905511811023622
Classification Report:

	precision	recall	f1-score	support
Apple___Apple_scab	0.91	0.55	0.68	126
Apple___Black_rot	0.90	0.90	0.90	124
Apple___Cedar_apple_rust	0.00	0.00	0.00	55
Apple___healthy	0.74	0.97	0.84	330
accuracy			0.79	635
macro avg	0.64	0.61	0.61	635
weighted avg	0.74	0.79	0.75	635

Fig 4.25 - Classification Report for Stacking

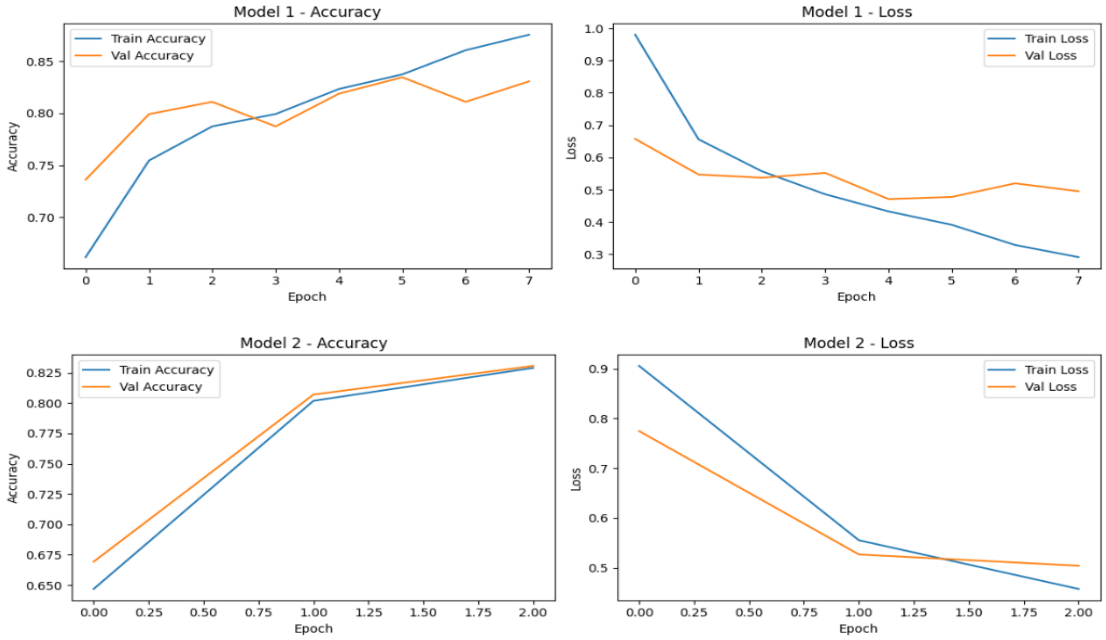


Fig 4.26 - Loss and Accuracy Curves for Voting

```
Voting Ensemble Accuracy: 0.8377952755905512
Classification Report:
                          precision    recall  f1-score   support

      Apple___Apple_scab       0.75      0.93      0.83       126
       Apple___Black_rot       0.80      0.95      0.87       124
Apple___Cedar_apple_rust       1.00      0.04      0.07        55
       Apple___healthy         0.90      0.89      0.90       330

                accuracy                           0.84       635
               macro avg       0.86      0.70      0.67       635
            weighted avg       0.86      0.84      0.81       635
```

Fig 4.27- Classification Report for Voting

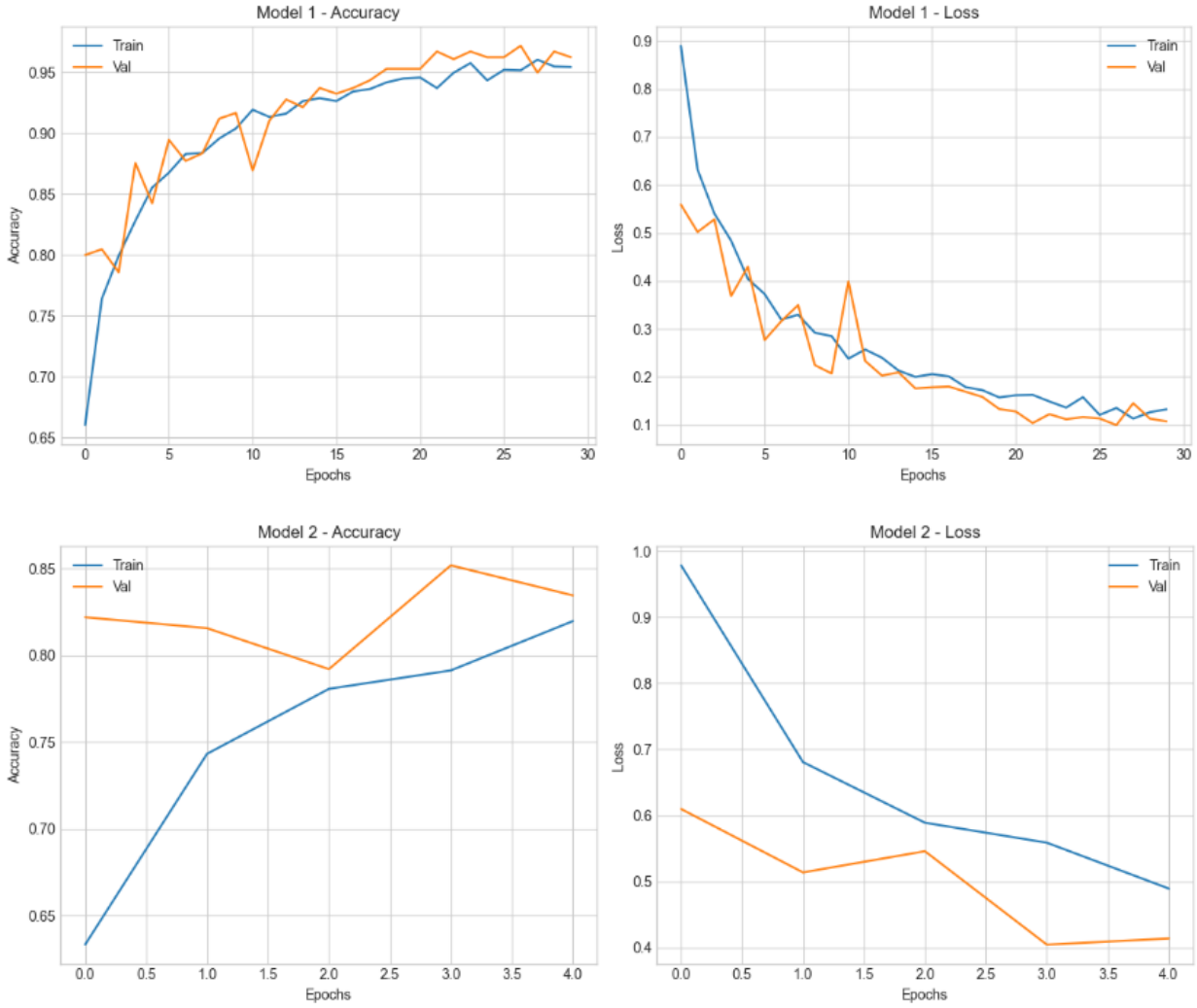


Fig 4.28 - Loss and Accuracy Curves for Blending

CHAPTER 5
CONCLUSION AND FUTURE PLANS

The presented work demonstrates the effective use of advanced deep learning architectures such as ResNet, Xception, DenseNet, and MobileNetV2 for the detection of apple plant diseases. These models were chosen due to their proven capability in extracting deep hierarchical features from images, which is essential in identifying subtle disease symptoms. The results showed high classification accuracy, indicating that these models are reliable tools for early detection of diseases affecting apple leaves.

In addition to standalone deep learning models, various ensemble learning techniques were applied, including bagging, boosting, stacking, voting, and blending. These techniques were integrated through sequential modeling approaches, which allowed the combination of multiple model predictions to improve overall performance. The ensemble methods effectively reduced overfitting and increased the robustness and precision of disease classification. As a result, the system significantly enhances early disease recognition, contributing to improved decision-making in precision agriculture and aiding farmers in timely crop management.

Looking ahead, the study can be further enhanced by adopting cutting-edge architectures like EfficientNet and Vision Transformers, which offer improved performance with optimized computational efficiency. Additionally, incorporating semi-supervised learning techniques would allow the model to learn from both labeled and unlabeled data, which is highly beneficial in agricultural datasets where annotation is costly and time-consuming.

Another promising direction is the development of real-time mobile applications. By deploying the trained models on smartphones or edge devices, farmers can detect diseases in the field without needing expert assistance or internet connectivity. Lastly, expanding the dataset to include images from diverse geographic locations, varying lighting conditions, and different stages of disease progression can further improve the model's generalization and accuracy. This would ensure the system remains effective under real-world conditions and across various environmental settings.

CHAPTER 6

REFERENCES

1. Vishnoi et al., "Detection of Apple Plant Diseases Using Leaf Images Through CNN," 2022.
2. Chen & Qi, "Identification of Plant Leaf Diseases by Deep Learning Based on Channel Attention and Channel Pruning," 2022.
3. Shorten & Khoshgoftaar, "Automatic and Reliable Leaf Disease Detection Using Deep Learning Techniques," 2021.
4. Mohanty et al., "Using Deep Learning for Image-Based Plant Disease Detection," Frontiers Plant Sci., 2016.
5. Calabro et al., "A Deep Learning Approach for Detection and Localization of Leaf Anomalies," 2022.
6. Yao et al., "Deep Learning for Plant Identification and Disease Classification from Leaf Images," 2023.
7. Krishnan et al., "Deep Learning Approaches for Image Recognition and Classification," 2022.
8. Shorten & Khoshgoftaar, "Data Augmentation for Improving Deep Learning in Image Classification Problems," 2021.

CHAPTER 7

APPENDIX

BASE PAPER

VIBHOR KUMAR VISHNOI , KRISHAN KUMAR , BRAJESH KUMAR ,SHASHANK MOHAN AND ARFAT AHMAD KHAN , "Detection of Apple Plant Diseases Using Leaf Images Through Convolutional Neural Network, " in IEEE Access (Volume: 11), pp. 109477-109487, 2023

DOI: 10.1109/ACCESS.2023.3322587.

Keywords: {Apple diseases, classification, convolutional neural network, deep learning, disease detection, image processing, machine learning }

URL: https://ieeexplore.ieee.org/document/10002365

SOURCE CODE

Importing Libraries

```
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import keras
from keras.callbacks import EarlyStopping,ModelCheckpoint
import tensorflow as tf
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
```

Calculating number of images and classes

```
color_path = r'C:\Users\kowsh\Desktop\main project\dataset\plantVillageDataset\color'
train_color = data(color_path)
print(train_color.shape)
train_color.label.value_counts().to_frame()
num_classes = train_color['label'].nunique()
print(f"Number of classes: {num_classes}")
```

Visualizing the image from each class

```
import os
import cv2
```

```
import numpy as np
import matplotlib.pyplot as plt
class_names = [dirname for dirname in os.listdir(color_path) if
os.path.isdir(os.path.join(color_path, dirname))]
plt.style.use('seaborn-v0_8-whitegrid')
plt.figure(figsize=(20, 20))
# Loop through each class
for idx, class_name in enumerate(class_names):
    # Get the path to the class directory
    class_dir = os.path.join(color_path, class_name)
    # List all images in the class directory
    image_files = [file for file in os.listdir(class_dir) if file.endswith(('.png', '.jpg', 'JPG', '.jpeg'))]
    if not image_files:
        continue  # Skip if no images are found in the class folder
    # Randomly select an image
    random_image = np.random.choice(image_files)
    # Load the image
    img_path = os.path.join(class_dir, random_image)
    img = cv2.imread(img_path)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    # Plot the image
    plt.subplot(len(class_names) // 5 + 1, 5, idx + 1)
    plt.imshow(img)
    plt.axis('off')
    plt.title(class_name, fontsize=15)
plt.tight_layout()
plt.show()
```

Data Augmentation & Normalization

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
# Define image size and batch size
image_size = (224, 224)
batch_size = 32
# Base ImageDataGenerator for rescaling
datagen = ImageDataGenerator(rescale=1./255)
# Train ImageDataGenerator with Data Augmentation
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
```

```
    horizontal_flip=True,
    fill_mode='nearest'
)
```

Data Collection From Each Directory

```
train_generator = train_datagen.flow_from_dataframe(
    dataframe=df_train,
    x_col='image',
    y_col='label',
    target_size=image_size,
    batch_size=batch_size,
    shuffle=True,
    class_mode='categorical'  # Ensure correct class_mode for multi-class problems
)

test_generator = datagen.flow_from_dataframe(
    dataframe=df_test,
    x_col='image',
    y_col='label',
    target_size=image_size,
    batch_size=batch_size,
    shuffle=False,
    class_mode='categorical'  # Ensure correct class_mode for multi-class problems
)

val_generator = datagen.flow_from_dataframe(
    dataframe=df_val,
    x_col='image',
    y_col='label',
    target_size=image_size,
    batch_size=batch_size,
    shuffle=False,
    class_mode='categorical'  # Ensure correct class_mode for multi-class problems
)
```

Building VGGNET Model

```
base_model = VGG16(weights="imagenet", include_top=False, input_shape=(224, 224, 3))

for layer in base_model.layers:
    layer.trainable = False
model = Sequential([
    base_model,  # Pretrained VGG16 feature extractor
    Flatten(),  # Flatten feature maps
    Dense(256, activation='relu'),  # Fully Connected Layer
    Dropout(0.3),  # Dropout to prevent overfitting
```

```
    Dense(4, activation='softmax')  # Output Layer (4 Classes)
])
model.compile(
    loss="categorical_crossentropy",
    optimizer=Adam(learning_rate=0.0001),
    metrics=["accuracy"]
)
```

Creating Callbacks

```
callbacks = [
    EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True, verbose=1),
    ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=2, verbose=1),
    ModelCheckpoint("best_model.h5", monitor='val_accuracy', save_best_only=True,
verbose=1)
]
```

Training the Model

```
epochs = 20
history = model.fit(
    train_generator,
    validation_data=val_generator,
    epochs=epochs,
    callbacks=callbacks
)

model.save("vgg16_model.h5")
from tensorflow.keras.models import load_model
```

Testing the model

```
best_model = load_model("best_model.h5")
test_loss, test_acc = best_model.evaluate(test_generator)
print(f"Test Accuracy of Best Model: {test_acc:.4f}")
```

Plotting accuracy and loss curves for Training and Validation

```
plt.style.use('seaborn-v0_8-whitegrid')  # Choose a clean and light background
plt.figure(figsize=(18, 6), facecolor='whitesmoke')  # Light figure background

# Loss Curve
plt.subplot(1, 3, 1, facecolor='floralwhite')  # Subplot background
plt.plot(history.history['loss'], 'b-', label='Training Loss')
plt.plot(history.history['val_loss'], 'r-', label='Validation Loss')
plt.title('Loss Curve', fontsize=14)
plt.xlabel('Epochs', fontsize=12)
plt.ylabel('Loss', fontsize=12)
plt.legend()

# Accuracy Curve
plt.subplot(1, 3, 2, facecolor='floralwhite')
plt.plot(history.history['accuracy'], 'b-', label='Training Accuracy')
plt.plot(history.history['val_accuracy'], 'r-', label='Validation Accuracy')
plt.title('Accuracy Curve', fontsize=14)
plt.xlabel('Epochs', fontsize=12)
plt.ylabel('Accuracy', fontsize=12)
plt.legend()

# Adjust layout and show the plot
plt.tight_layout(pad=2.0)  # Adds padding between subplots
plt.show()
```

Confusion Matrix & Classification Report

```
y_pred = np.argmax(model.predict(test_generator), axis=1)
y_true = test_generator.classes
class_labels = list(test_generator.class_indices.keys())

CM = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(15, 25))
sns.heatmap(CM, fmt='g', center=True, cbar=False, annot=True, cmap='Set2',
        xticklabels=class_labels, yticklabels=class_labels)
plt.title('Confusion Matrix')
plt.show()

classification_report_str = classification_report(y_true, y_pred, target_names=class_labels)
print('Classification Report:\n', classification_report_str)
```

Predicting output class for the given image

```
image_path = r"C:\Users\kowsh\Desktop\main
project\dataset\plantVillageDataset\color\BlackRot\1a9f6dfb-fdf4-43b4-9fb4-
b7a809b49b9d___JR_FrgE.S 2765.JPG"
input_image = preprocess_image(image_path)

# Predict
predictions = model.predict(input_image)
predicted_class = np.argmax(predictions)  # Get class with highest probability

# Print Results
predicted_label = class_labels[predicted_class]  # Map to disease name

print(f"Predicted Disease: {predicted_label}")

#testing 2
image_path = r"C:\Users\kowsh\Desktop\main
project\dataset\plantVillageDataset\color\Healthy\0e8c4b90-36d4-4b8c-a134-
88c344e97ac2___RS_HL 5721.JPG"
input_image = preprocess_image(image_path)
predictions = model.predict(input_image)
predicted_class = np.argmax(predictions)
predicted_label = class_labels[predicted_class]
print(f"Predicted Disease: {predicted_label}")
```

Building ResNet50 Model

```
base_model = ResNet50(weights="imagenet", include_top=False, input_shape=(224, 224, 3))

for layer in base_model.layers:
    layer.trainable = False
model = Sequential([
    base_model,  # Pretrained Resnet152 feature extractor
    Flatten(),  # Flatten feature maps
    Dense(256, activation='relu'),  # Fully Connected Layer
    Dropout(0.3),  # Dropout to prevent overfitting
    Dense(4, activation='softmax')  # Output Layer (4 Classes)
])
```

Building ResNet512 Model

```
base_model = ResNet512(weights="imagenet", include_top=False, input_shape=(224, 224, 3))

for layer in base_model.layers:
    layer.trainable = False
model = Sequential([
    base_model,  # Pretrained Resnet152 feature extractor
    Flatten(),  # Flatten feature maps
    Dense(256, activation='relu'),  # Fully Connected Layer
    Dropout(0.3),  # Dropout to prevent overfitting
    Dense(4, activation='softmax')  # Output Layer (4 Classes)
])
```

Building InceptionV3 Model

```
base_model = InceptionV3(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
for layer in base_model.layers:
    layer.trainable = False
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(256, activation='relu')(x)
x = Dropout(0.5)(x)
num_classes = 4
output_layer = Dense(num_classes, activation='softmax')(x)
model = Model(inputs=base_model.input, outputs=output_layer)
model.compile(optimizer=Adam(learning_rate=0.0001), loss='categorical_crossentropy',
metrics=['accuracy'])
model.fit(train_generator, validation_data=val_generator, epochs=10)
model.save("inception_model.h5")
```

Building Xception Model

```
base_model = Xception(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
for layer in base_model.layers:
    layer.trainable = False
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(256, activation='relu')(x)
x = Dropout(0.5)(x)
num_classes = 4
output_layer = Dense(num_classes, activation='softmax')(x)
model = Model(inputs=base_model.input, outputs=output_layer)
```

```
model.compile(optimizer=Adam(learning_rate=0.0001), loss='categorical_crossentropy',
metrics=['accuracy'])
model.fit(train_generator, validation_data=val_generator, epochs=10)
model.save("inception_model.h5")
```

Building DenseNet201 Model

```
base_model = DenseNet201(weights="imagenet", include_top=False, input_shape=(224, 224,
3))

for layer in base_model.layers:
    layer.trainable = False

model = Sequential([
    base_model,  # Pretrained Resnet152 feature extractor
    Flatten(),  # Flatten feature maps
    Dense(256, activation='relu'),  # Fully Connected Layer
    Dropout(0.3),  # Dropout to prevent overfitting
    Dense(4, activation='softmax')  # Output Layer (4 Classes)
])
model.compile(
    loss="categorical_crossentropy",
    optimizer=Adam(learning_rate=0.0001),
    metrics=["accuracy"]
)
```

Building MobileNetV2 Model

```
base_model = MobileNetV2(weights="imagenet", include_top=False, input_shape=(224, 224,
3))

for layer in base_model.layers:
    layer.trainable = False
model = Sequential([
    base_model,  # Pretrained VGG16 feature extractor
    Flatten(),  # Flatten feature maps
    Dense(256, activation='relu'),  # Fully Connected Layer
    Dropout(0.3),  # Dropout to prevent overfitting
    Dense(4, activation='softmax')  # Output Layer (4 Classes)
])
```

Building Sequential Models for Ensembling Methods

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.optimizers import Adam
```

```
def create_model1():
    model = Sequential([
        Conv2D(32, (3, 3), activation='relu', input_shape=(IMG_SIZE, IMG_SIZE, 3)),
        MaxPooling2D(2, 2),
        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D(2, 2),
        Flatten(),
        Dense(64, activation='relu'),
        Dropout(0.3),
        Dense(len(np.unique(y_train)), activation='softmax')
    ])
    model.compile(optimizer=Adam(0.001), loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
    return model

def create_model2():
    model = Sequential([
        Conv2D(64, (3, 3), activation='relu', input_shape=(IMG_SIZE, IMG_SIZE, 3)),
        MaxPooling2D(2, 2),
        Conv2D(128, (3, 3), activation='relu'),
        MaxPooling2D(2, 2),
        Flatten(),
        Dense(128, activation='relu'),
        Dropout(0.4),
        Dense(len(np.unique(y_train)), activation='softmax')
    ])

def create_model3():
    model = Sequential([
        Conv2D(128, (3, 3), activation='relu', input_shape=(IMG_SIZE, IMG_SIZE, 3)),
        MaxPooling2D(2, 2),
        Dropout(0.3)
        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D(2, 2),
        Flatten(),
        Dense(256, activation='relu'),
        Dropout(0.4),
        Dense(len(np.unique(y_train)), activation='softmax')
    ])
    model.compile(optimizer=Adam(0.001), loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
    return model
```

Building Bagging Method

```
pred1 = model1.predict(X_test)
pred2 = model2.predict(X_test)
pred3 = model3.predict(X_test)

# Bagging by averaging the predictions
final_pred = (pred1 + pred2 + pred3) / 3
final_labels = np.argmax(final_pred, axis=1)
true_labels = np.argmax(y_test, axis=1)

print("Bagging Ensemble Accuracy:", accuracy_score(true_labels, final_labels))
print("Classification Report:\n", classification_report(true_labels, final_labels,
target_names=le.classes_))
```

Building Boosting Model

```
pred1 = model1.predict(X_test)
pred2 = model2.predict(X_test)
pred3 = model3.predict(X_test)

# Weighted ensemble (Boosting-style)
final_pred = (0.2 * pred1) + (0.3 * pred2) + (0.5 * pred3)
final_labels = np.argmax(final_pred, axis=1)
true_labels = np.argmax(y_test, axis=1)

print("Boosting Ensemble Accuracy:", accuracy_score(true_labels, final_labels))
print("Classification Report:\n", classification_report(true_labels, final_labels,
target_names=le.classes_))
```

Building Stacking Model

```
pred1_train = model1.predict(X_train)
pred2_train = model2.predict(X_train)
pred3_train = model3.predict(X_train)

meta_X_train = np.concatenate([pred1_train, pred2_train, pred3_train], axis=1)

meta_input = Input(shape=(meta_X_train.shape[1],))
x = Dense(64, activation='relu')(meta_input)
x = Dropout(0.3)(x)
x = Dense(32, activation='relu')(x)
meta_output = Dense(y.shape[1], activation='softmax')(x)
meta_model = Model(meta_input, meta_output)
```

```
meta_model.compile(optimizer=Adam(learning_rate=0.0005), loss='categorical_crossentropy',
metrics=['accuracy'])
meta_model.fit(meta_X_train, y_train, epochs=25, batch_size=32, validation_split=0.15,
callbacks=callbacks)
```

Building Voting Model

```
pred1 = model1.predict(X_test)
pred2 = model2.predict(X_test)
final_pred = (pred1 + pred2) / 2
final_labels = np.argmax(final_pred, axis=1)
true_labels = np.argmax(y_test, axis=1)

print("Voting Ensemble Accuracy:", accuracy_score(true_labels, final_labels))
print("Classification Report:\n", classification_report(true_labels, final_labels,
target_names=le.classes_))
```

Building Blending Model

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Get softmax outputs for validation set
pred1 = model1.predict(X_val)
pred2 = model2.predict(X_val)

# Stack them horizontally to form meta features
meta_features = np.concatenate([pred1, pred2], axis=1)

# Train logistic regression as meta-classifier
meta_model = LogisticRegression(max_iter=1000)
meta_model.fit(meta_features, y_val)

# Final blended predictions
final_preds = meta_model.predict(meta_features)

# Accuracy
blended_accuracy = accuracy_score(y_val, final_preds)
print(f"Blended Ensemble Accuracy: {blended_accuracy * 100:.2f}%")
```