Comparative Analysis of Fruits Disease Detection using Different types of CNN Architecture

Comilla University

This Project Report is Submitted as a Partial Requirement for the Completion of the M.Sc. Engineering , in Information and Communication Technology

Submitted By:

ID NO:22209014 ID NO:22209034

Session: 2021-2022

Department of Information and Communication Technology

Supervised By:

Dr. Dulal Chakraborty

Associate Professor

Department of Information and Communication Technology

Faculty OF Engineering

Comilla University, Bangladesh

February, 2025

Acknowledgement

In the name of Allah, the Most Merciful, the Most Gracious, Alhamdulillah, all praise and gratitude are due to Allah Almighty for granting me the strength, patience, and knowledge to complete this project successfully.

First and foremost, we would like to express our sincere gratitude to my supervisor, **Dr. Dulal Chakraborty Sir,** for his continuous guidance, insightful feedback, and invaluable support throughout the duration of this project. His expertise and encouragement have been instrumental in shaping the direction of our research.

We are deeply thankful to the **Department of ICT, Comilla University**, for providing the opportunity and resources to undertake this project, as well as for fostering a supportive academic environment.

Our heartfelt thanks also go to our family and friends for their unwavering encouragement, patience, and understanding during this challenging journey. Their support has been our source of strength.

Finally, We are grateful to everyone who contributed to this project in any capacity. Your inspiration, suggestions, and collaboration were vital to the successful completion of this research.

Abstract

Fruit diseases faces significant challenges to agricultural productivity, leading to substantial economic losses globally. Early and accurate detection of these diseases is crucial for effective management and prevention. In recent years, Deep Learning, particularly Convolutional Neural Networks (CNNs), has emerged as a powerful tool for image classification and disease detection in agriculture.

This project focuses on a comparative analysis of various CNN architectures for detecting diseases in fruits such as guava, mango, and orange. The study involves seven specific fruit diseases, including anthracnose in guava and mango, black mould rot in mango, black spot in orange, greening in orange, fruit fly in guava, and stem rot in mango. A dataset comprising 11 classes with 5,091 images was utilized to train and evaluate six CNN architectures: a 5-layer CNN, LeNet, VGG16, MobileNet, Inception V3, and ResNet.

The experimental results reveal that Inception V3 achieved the highest test accuracy of 96.71%, followed by VGG16 with 93.41% and LeNet with 90.47%. MobileNet, ResNet, and the 5-layer CNN recorded accuracies of 88.21%, 88.91%, and 84.06%, respectively. These findings highlight the superior performance of Inception V3 for fruit disease detection tasks and underline the potential of deep learning in developing efficient and automated solutions for agricultural applications.

This research contributes to the field by identifying the strengths and limitations of different CNN architectures, providing valuable insights for future advancements in fruit disease detection systems.

Index

Title **Page No.**

Table of Contents

Chapter Title **Page No.**

List of Figures

Figure Name **Page No.**

List of tables

Chapter 1

Introduction

1.1 INTRODUCTION

Agriculture remains the backbone of Bangladesh's economy, with a significant portion of the population relying on crop production for sustenance and income. Fruits such as guava, orange, and mango are vital to the agricultural sector, playing a key role in both food security and economic stability. However, these fruits are increasingly susceptible to various diseases, which can lead to reduced yield, lower quality, and financial losses for farmers. Among these, anthracnose in guava, anthracnose in mango, Black mould rot in mango, black spot in orange, fruit fly in guava, Greening in Orange**,** and Stem Rot in Mango are some of the most pressing challenges. These diseases, caused by biotic factors like fungi, bacteria, and viruses, or abiotic factors such as extreme weather and nutrient deficiencies, manifest through visible symptoms on the fruit's surface, such as discoloration, spots, and lesions. Timely and accurate detection of these diseases is critical for effective management and to prevent widespread crop damage.

Traditionally, the detection of these diseases has been carried out manually, requiring experts to visually inspect the fruits. However, this method is not only time-consuming and labor-intensive but also prone to human error due to the complexity and variability of disease symptoms. Advancements in artificial intelligence (AI) and machine learning have led to the rise of deep learning models, especially Convolutional Neural Networks (CNNs), as a robust solution for automated disease detection using images. CNNs can autonomously extract features from images and classify them with remarkable accuracy, making them highly effective for identifying diseases in fruits.

This study focuses on performing a comparative analysis of fruit disease detection using six CNN architectures: 5-layer CNN, LeNet, VGG16, MobileNet, Inception V3, and ResNet. By leveraging these models, the research aims to classify diseases affecting guava, orange, and mango fruits, including anthracnose, Black mould rot, black spot , fruit fly , Greening and Stem Rot**.** The primary objective is to evaluate the performance of these architectures in terms of accuracy, precision, recall, and other performance metrics. The results will identify the most efficient model for fruit disease detection, which can empower farmers to take timely actions and mitigate losses in crop yield.

The study will not only automate the disease detection process but will also empower farmers to identify infections in their crops at an early stage. This is particularly useful in regions where farmers lack access to advanced diagnostic tools and rely on traditional methods. By offering an affordable and reliable solution, this research contributes to improving productivity and ensuring food security in Bangladesh and similar agricultural-based economies.

1.2 Motivation

The motivation for this work lies in the need for early detection and classification of fruit

diseases using advanced deep learning techniques. Traditional manual methods for identifying fruit diseases are not only time-consuming and prone to errors but also heavily reliant on expert knowledge, limiting their scalability and efficiency. Given the critical role fruits play in global agriculture, the ability to detect diseases early can significantly minimize economic losses and enhance food security.

Artificial intelligence (AI) and computer vision offer innovative solutions to address the limitations of manual inspection. In particular, Deep Neural Networks (DNNs), particularly Convolutional Neural Networks (CNNs), have shown exceptional performance in image classification tasks,making them an ideal choice for detecting fruit diseases. CNNs are capable of learning hierarchical visual features and generalizing across diverse datasets, enabling them to identify complex patterns in fruit images.

This research focuses on conducting a comparative analysis of different CNN architectures, including LeNet, VGG16, MobileNet, Inception V3, ResNet, and a custom 5-layer CNN, to evaluate their performance in fruit disease detection. By applying these architectures to datasets containing diseases in fruits such as guava, mango, and orange, this study aims to identify the most effective model for accurate and efficient disease classification.

The outcome of this project will contribute to developing automated, scalable, and accurate systems for fruit disease detection. Such systems can empower farmers to take timely action, reduce losses, and improve both the quality and quantity of fruit production. The findings will also provide valuable insights for optimizing CNN architectures in agricultural applications, paving the way for further advancements in precision farming.

1.3 Impact of Fruit Diseases on Agriculture and the Economy

Fruit Diseases present a major threat to global agricultural productivity, causing severe economic losses for farmers and disrupting food supply chains. These diseases affect both the quantity and quality of fruit crops, leading to reduced yields and sometimes entire crop failures. Fruits such as guava, mango, and orange are susceptible to a wide variety of diseases, including fungal, bacterial, and viral infections, which can spread rapidly if not detected and controlled at an early stage.

The impact on agriculture is multifaceted, as these diseases often result in increased production costs, including the need for additional pesticides, labor for manual inspection, and crop replacement. For farmers, the loss of a significant portion of their produce can lead to substantial financial hardships. Moreover, fruit diseases also affect consumers by leading to higher market prices, reduced availability, and lower-quality fruits.

Economically, fruit diseases can lead to reduced export opportunities, as many countries rely on the international trade of fruits for revenue generation. For instance, tropical fruit-exporting countries such as Bangladesh, India, and Brazil can experience considerable losses when disease outbreaks affect export-quality fruits. Thus, the timely detection and management of fruit diseases are critical for maintaining a stable agricultural economy and ensuring the livelihood of millions of farmers.

Early detection methods, such as those enabled by deep learning techniques, offer the potential to significantly mitigate these challenges by providing accurate and rapid diagnosis.

This can empower farmers to take preventive measures before the diseases cause irreversible damage, ultimately improving crop yield, quality, and profitability.

1.4 Limitation of Fruit Disease Detection Using CNN Architectures

Despite its advantages, the project faces several limitations and challenges:

Dataset Quality and Quantity**:** The performance of CNN models is heavily reliant on the availability of diverse and high-quality datasets. Insufficient or imbalanced data can lead to biased predictions.

Model Generalization: Models trained on particular datasets may have difficulty generalizing to new or unseen data to real-world scenarios due to variations in lighting, background, or fruit appearance.

Computational Requirements**:** High-performing models like ResNet and Inception V3 require significant computational resources, which might not be feasible for small-scale farmers or resource-limited environments.

Energy Consumption: Training and deploying deep CNNs can be energy-intensive, raising concerns about sustainability in resource-constrained settings.

Real-Time Processing Challenges**:** Achieving real-time disease detection in field conditions remains challenging due to hardware limitations and environmental factors like lighting and occlusion.

Integration with Field Systems**:** Deploying the system in real agricultural settings, such as embedding it in drones or IoT devices, involves technical and logistical complexities.

Disease Overlap: Visual symptoms of different diseases may overlap, complicating accurate classification without additional contextual data.

1.5 Aims and Objectives

The main objective of this research is to investigate and compare the effectiveness of various Convolutional Neural Network (CNN) architectures in detecting and classifying fruit diseases with the goal of enhancing agricultural disease detection systems for improved crop yield, quality, and overall productivity.

Objectives:

1. To compare the performance of six CNN architectures—5-layer CNN, LeNet, VGG16, MobileNet, Inception V3, and ResNet—in classifying diseases in fruits such as guava, mango, and orange.
2. To evaluate and analyze the accuracy and efficiency of each CNN model in detecting fruit diseases, focusing on identifying the most suitable architecture for early disease detection.
3. To optimize the deep learning models for fruit disease detection by exploring various hyperparameters and training techniques, aiming to achieve the highest possible accuracy and efficiency.

4. To investigate the impact of different CNN architectures on the detection of specific fruit diseases, including anthracnose in guava, black spot in orange, and stem rot in mango.
5. To propose an automated fruit disease detection system that can assist farmers in identifying plant diseases early, thereby helping them take preventive measures to reduce crop loss.
6. To contribute valuable insights into the challenges and best practices of using deep learning in agricultural applications, providing recommendations for future work in the area of precision farming and disease management.

1.6 Outline

Chapter 1: Introduction

- This chapter introduces the research problem, outlining the importance of fruit disease detection and the role of deep learning in solving this issue.
- The chapter includes the problem statement, objectives, and aims of the project.
- An overview of the challenges in fruit disease detection and the motivation for using CNNs is presented.

Chapter 2: Literature Review

- This chapter examines existing studies on fruit disease detection through deep learning methods, particularly emphasizing Convolutional Neural Networks (CNNs).
- It highlights the evolution of CNNs in agricultural applications and discusses the current state of research in fruit disease classification.
- The chapter also compares different CNN architectures used in previous studies and their results.

Chapter 3: Methodology

- This chapter describes the methodology followed in this study, including data collection, preprocessing, and model selection.
- It provides a detailed explanation of the CNN architectures used: 5-layer CNN, LeNet, VGG16, MobileNet, Inception V3, and ResNet.
- The hardware and software requirements for model development, training, and evaluation are discussed.

Chapter 4: Results and Analysis

- This chapter presents the results of the CNN models in terms of accuracy, precision, recall, and other performance metrics.
- The chapter also includes visualizations, such as confusion matrices and training/testing accuracy graphs.
- A comparative analysis of the models is conducted to identify the most efficient architecture for fruit disease detection.

Chapter 5: Discussion

- In this chapter, the implications of the results are discussed in relation to the agricultural impact of fruit disease detection.
- The strengths and limitations of each CNN model are highlighted, along with the challenges faced during the study.
- Recommendations for improving future models and overcoming current challenges are provided.

Chapter 6: Conclusion and Future Work

- This chapter summarizes the key findings of the research and discusses the potential real-world applications of the proposed system.
- It also suggests possible directions for future research, including the inclusion of more diverse datasets, hybrid models, and integration with IoT-based farming solutions.

CHAPTER 2

Background

2.1 Literature Review

The effectiveness of convolutional neural networks (CNNs) in fruit disease detection has been explored extensively in recent years. This chapter reviews various studies related to the performance evaluation of CNN architectures in this domain, focusing on their methodologies, datasets, and performance metrics.

The author Mohanty et al. in "Using Deep Learning for Image-Based Plant Disease Detection" explored deep learning for plant disease identification and set a foundation for its application in fruit disease detection. Their use of AlexNet and GoogLeNet architectures demonstrated the superior accuracy of CNNs compared to traditional methods. Despite focusing on plant leaves, their methodology has inspired numerous studies on fruit-specific disease detection.

The author LeCun et al. in "Gradient-Based Learning Applied to Document Recognition" introduced LeNet, a simple CNN architecture with two convolutional layers followed by pooling and fully connected layers. It has been used as a benchmark in studies evaluating the performance of lightweight models on fruit disease datasets.

The author Simonyan and Zisserman in "Very Deep Convolutional Networks for Large-Scale Image Recognition" proposed VGG16, a deep CNN with 16 layers. Its uniform architecture has proven effective in fruit disease detection studies, such as those focusing on anthracnose in mango and citrus greening in oranges. VGG16's depth enables it to capture complex patterns in large datasets.

The author Howard et al. in "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications" developed MobileNet, a lightweight CNN optimized for mobile and embedded applications. Its depthwise separable convolutions reduce computational costs, making it a popular choice for resource-constrained environments. Studies have highlighted its balance between accuracy and efficiency in fruit disease detection tasks.

The author Szegedy et al. in "Rethinking the Inception Architecture for Computer Vision" introduced Inception V3, which uses inception modules to process features at multiple scales within a single layer. This architecture has been employed in detecting diseases such as black spot in oranges and fruit fly damage in guavas, demonstrating its ability to capture fine-grained features.

The author He et al. in "Deep Residual Learning for Image Recognition" proposed ResNet, which addresses the vanishing gradient problem through residual learning. Its skip connections make it suitable for deep learning tasks involving complex datasets. ResNet has

been shown to excel in detecting diseases like stem rot in mangoes and anthracnose in guavas, achieving high accuracy on diverse datasets.

The author of custom-designed models introduced a 5-layer CNN to evaluate its performance against standard architectures. This approach allows for flexibility in design and optimization for fruit disease datasets.

Publicly available datasets such as PlantVillage are commonly used for training and testing CNN models in fruit disease detection. However, specific datasets for diseases like anthracnose in mango and guava, black spot in oranges, and fruit fly damage in guava remain limited. Data augmentation techniques, including rotation, flipping, and noise addition, are frequently employed to enhance the diversity of training datasets.

Performance metrics like accuracy, precision, recall, and F1-score are used to evaluate the model's effectiveness widely used to evaluate CNN models. Additionally, metrics like confusion matrices provide insights into classification performance for individual disease categories. Computational efficiency metrics, including model size and inference time, are crucial when deploying models on resource-constrained devices.

Comparative analyses of CNN architectures have highlighted their strengths and weaknesses in fruit disease detection. For instance, MobileNet is noted for its efficiency, while ResNet achieves superior accuracy for large-scale datasets. Studies have also explored the trade-offs between accuracy and computational requirements, emphasizing the importance of context-specific architecture selection.

The reviewed literature underscores the effectiveness of CNNs in automating fruit disease detection, with architectures like ResNet and MobileNet emerging as top performers. This study builds on these findings by conducting a comparative analysis of six CNN architectures—LeNet, VGG16, MobileNet, Inception V3, ResNet, and a custom 5-layer CNN—to identify the optimal approach for detecting diseases in guava, orange, and mango fruits.

2.2Types of Fruit Diseases

Fruit diseases are broadly categorized based on their causative agents. These diseases can be abiotic, caused by non-living environmental factors, or biotic, caused by living organisms such as fungi, bacteria, viruses, nematodes, and pests. Biotic diseases, particularly those affecting fruits, pose significant threats to crop yield and quality.

Biotic Diseases

Biotic fruit diseases occur due to interactions between fruits and living organisms. These diseases are often difficult to manage and can cause severe damage to crops, including fruits. Understanding these diseases is crucial for fruit disease detection and management, especially in the context of this research.

Fungal Diseases

Fungal diseases are one of the primary causes of fruit crop damage. Fungal pathogens infect fruits directly, leading to loss of yield and quality. These diseases spread through spores that

can be dispersed via air, water, and contact. Some common fungal diseases affecting fruits include:

Anthracnose**:** Causes black or brown spots on fruits, often spreading to stems and leaves.

Powdery Mildew: Leads to white, powdery patches on the surface of fruits, affecting the overall quality.

Apple Scab**:** Results in dark, scabby lesions on apples, impacting fruit appearance and market value.

Gray Mold: Commonly found in strawberries, causing soft, mushy rot.

Verticillium Wilt: Affects many fruit trees, causing wilting and leaf discoloration.

Rusts: These affect fruit crops by creating spots and discoloration, leading to lower yield.

Bacterial Diseases
Bacterial infections in fruits spread quickly, especially through rain, irrigation water, or infected pruning tools. Bacteria invade plant tissues and multiply, leading to rotting and discoloration. Common bacterial diseases affecting fruits include:

Bacterial Canker**:** Common in stone fruits like cherries and peaches, causing lesions and cracking on fruits.

Bacterial Spot: Affects fruits such as tomatoes and stone fruits, leading to black or brown spots that reduce fruit marketability.

Fire Blight: Affects fruits like apples and pears, leading to wilting, browning of leaves, and rotting of fruits.

Crown Gall: Causes tumors on fruit trees at the base of the plant, reducing productivity.

Soft Rot: Common in fruits like potatoes and tomatoes, leading to rapid decay in stored crops.

Virus Diseases
Virus infections in fruits are transmitted by insects, contaminated tools, or infected planting materials. Viruses cause persistent damage, often leading to stunted growth, deformation, and poor fruit quality. Viral diseases are harder to control, making early detection crucial. Common viral diseases affecting fruits include:

Mosaic Virus: Leads to yellowing and streaking of leaves, and mottling of fruits like tomatoes and peppers.

Wilt Virus**:** Causes wilting in various fruit crops, leading to plant death.

Spotted Wilt Virus**:** Affects fruits like tomatoes and peppers, causing spots on the leaves and deformities in fruits.

Yellowing Virus**:** Causes yellowing in fruit crops, leading to nutrient deficiencies and reduced yield.

Grapevine Leafroll Virus: Affects grapevines, leading to leaf curling, discoloration, and reduced fruit production.

Impact on Fruits

Fungal, bacterial, and viral diseases significantly impact fruit crops by reducing yield, damaging fruit quality, and making fruits unmarketable. For example:

Anthracnose can result in large-scale damage to fruits like mangoes and guavas, leading to premature fruit drop.

Powdery Mildew affects fruits like apples and grapes, causing reduced photosynthesis and impacting sugar content.

Bacterial Canker leads to discoloration and sunken lesions on fruits, decreasing commercial value.

Fire Blight can decimate entire orchards by causing quick wilting and fruit rot.

Importance of Early Detection in Fruit Disease

In our project, focusing on **disease detection** in fruits, early detection is crucial to minimize losses. Timely identification of diseases through image processing, machine learning, and computer vision can help in:

Preventing Spread: Early detection can prevent the widespread spread of diseases within orchards.

Reducing Economic Losses: Minimizing the impact of infections on crop yields ensures better fruit quality and marketability.

Implementing Targeted Treatments: Accurate diagnosis through detection systems enables farmers to apply the right treatment at the right time.

2.3 General Plant Diseases Detection System

A General Plant Diseases Detection System is an application of technology to monitor and identify diseases affecting plants, typically by using various data sources like images, environmental data, and machine learning algorithms. These systems are especially valuable in agriculture and horticulture to help farmers and gardeners detect and manage diseases in their crops or plants early, potentially preventing widespread damage.

Key Components:

Data Collection:

Image Data: Use cameras (e.g., mobile phones, drones, or fixed cameras) to capture images of plants and their leaves, stems, flowers, or fruits.

Environmental Data: Information like humidity, temperature, soil moisture, and rainfall, which can help correlate conditions that may lead to diseases.

Image Processing:

Preprocessing: Enhancing and preparing the images for analysis (e.g., noise removal, image sharpening).

Feature Extraction: Identifying patterns or regions of interest within the image (e.g., spots, lesions, discoloration, deformations).

Disease Classification:

Machine Learning Models: Trained models such as Convolutional Neural Networks (CNNs) or other deep learning techniques for plant disease detection. These models can classify diseases based on visual symptoms.

Pattern Recognition: Identify specific symptoms of diseases (e.g., rust, blight, mildew, bacterial infections, etc.) by comparing input data to a pre-trained database of plant diseases.

Model Training: Using a dataset that includes labeled images of plants with diseases. Over time, the system improves as more images are added and the model is retrained.

Data Integration:

Agricultural Databases: Integration with existing agricultural databases for disease symptoms, treatment methods, and preventative strategies.

Real-time Analysis: Combining image and environmental data to offer real-time disease diagnosis.

Technologies Used:

Deep Learning: Using CNNs or transfer learning (e.g., pre-trained models like ResNet, VGGNet,Inceptionv3,Alexnet,LeNet,Mobailenet,5LayerCnn) for image classification tasks.

Computer Vision: For image processing and feature extraction from plant images.

IoT (Internet of Things): Sensors placed in fields or greenhouses to capture environmental data and correlate with disease detection.

Benefits:

Early Detection: Identifying diseases in their early stages can lead to more effective treatment and reduced crop loss.

Cost-Effective: Helps reduce the need for expert field visits, lowering diagnostic costs.

Sustainability: Enables the use of more targeted treatments, reducing the overuse of pesticides and minimizing environmental impact.

Real-time Insights: Allows farmers to act immediately, often preventing the spread of a disease across crops fruits.

Example Systems:

PlantVillage: A project that uses machine learning to identify plant diseases from images taken with smartphones.

Plantix: A mobile app that helps users diagnose plant diseases based on image recognition.

2.4 Evolution of CNN architectures

The evolution of Convolutional Neural Network (CNN) architectures is a fascinating journey that highlights how the field of deep learning has progressed over the years. CNNs have become the cornerstone of modern computer vision tasks, Due to their capacity to autonomously learn hierarchical features from raw image data. Below is an overview of the major milestones in the evolution of CNN architectures.

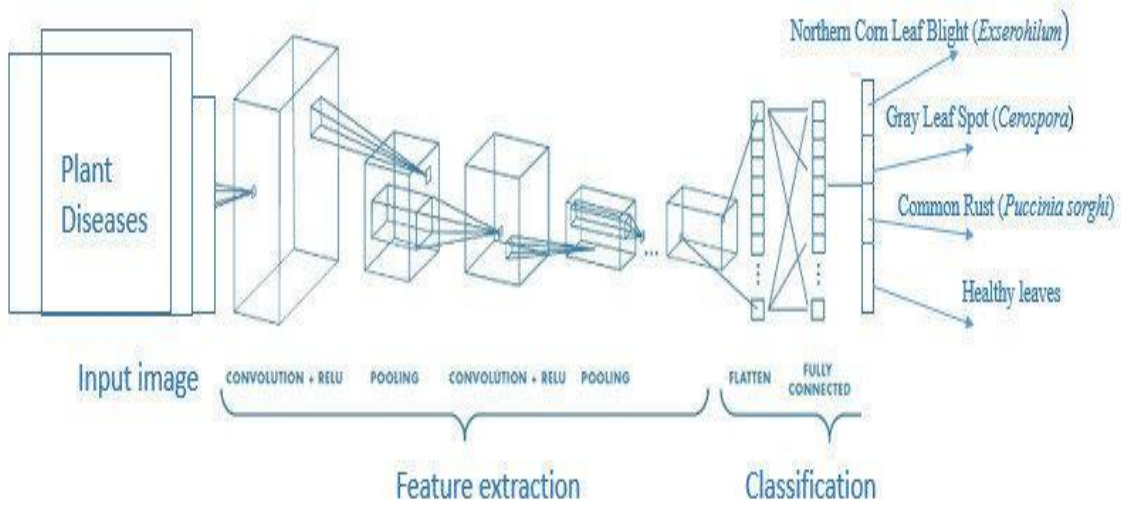


Fig 2.1: Feature extraction and classification process in CNN

Convolution Layer:

The convolutional layer in a Convolutional Neural Network (CNN) plays a crucial role in processing spatial data, such as images, by detecting local patterns and features. The use of machine learning (ML), specifically deep learning, in conjunction with convolutional layers

enables the network to automatically learn these features without explicit feature engineering. Below is an explanation of why convolutional layers use ML and how they function within a CNN.

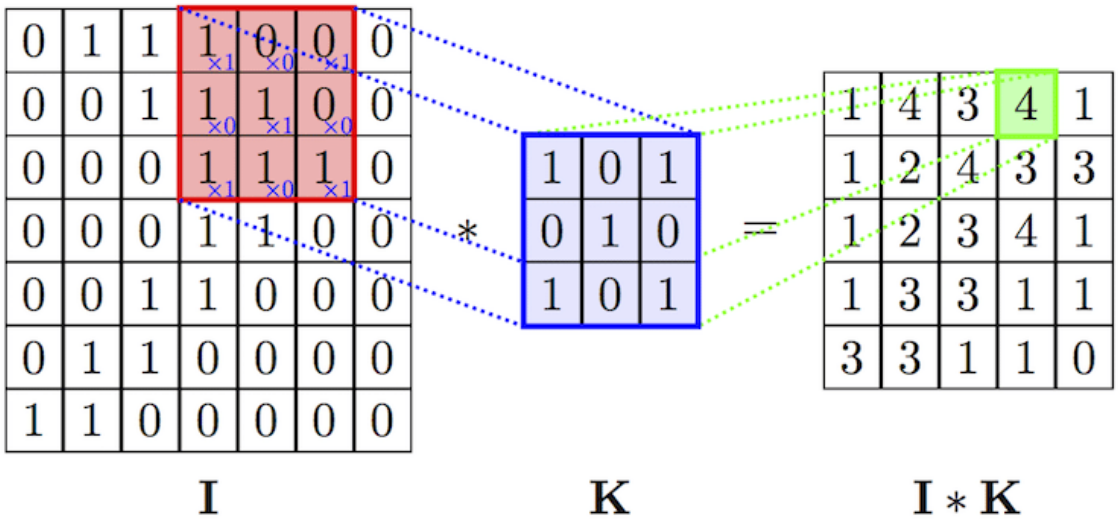


Fig2.2:Convolution Layer

Pooling Layer:

A pooling layer is typically used after a convolutional layer in a CNN. Its primary function is to downsample the spatial dimensions (height and width) of the feature maps generated by the convolutional layer. The pooling operation helps reduce the number of parameters, prevent overfitting, and improve computational efficiency while preserving the most important features in the image.

Types of Pooling:

Max Pooling: Takes the maximum value from a region of the feature map (typically a 2x2 or 3x3 window).

Average Pooling: Takes the average value from a region of the feature map.

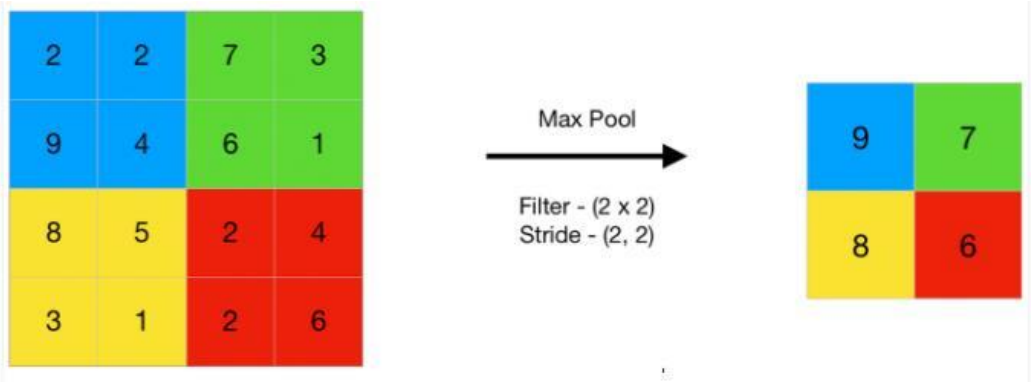


Fig2.3:Pooling Layer

Fully Connected Layer:

A **Fully Connected (FC) Layer** is a layer in which each neuron is connected to every neuron in the previous layer. In contrast to convolutional layers (which apply local filters to detect features), the fully connected layer combines all the extracted features from previous layers to make a final decision or prediction. The output from the last fully connected layer is typically passed through an activation function (like softmax for classification) to produce the final output of the network.

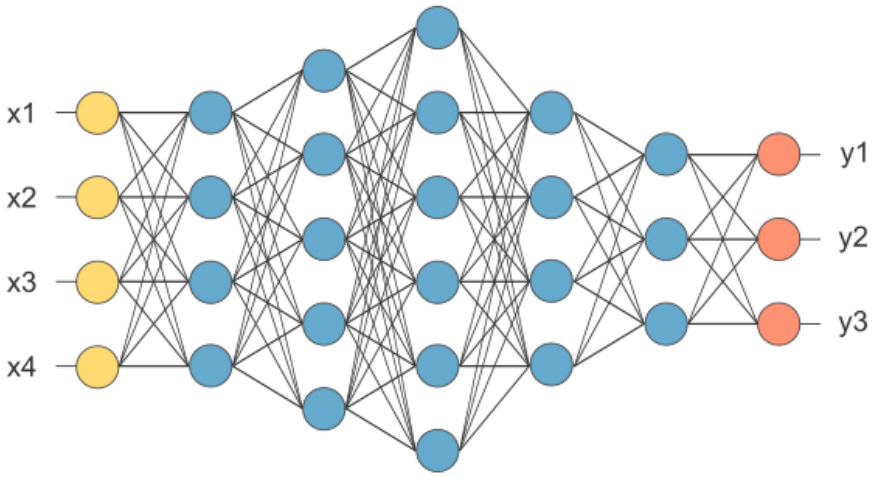


Fig2.4:Fully connected Layer

2.4.1 5-layer CNN

A 5-layer Convolutional Neural Network (CNN) architecture is a simple deep learning model used for image recognition tasks and other computer vision applications. In this architecture, the number of layers can be broken down into various types of layers that each serve a specific purpose (e.g., convolution, pooling, fully connected).

 Input Layer: The first layer is the input layer where the image data is fed into the network. For example, an input image could be of size 32×32×332 \times 32 \times 332×32×3, where 32x32 are the spatial dimensions and 3 corresponds to the RGB color channels.

- **Input Size**: 32×32×332 \times 32 \times 332×32×3
- **Description**: This is the raw image data in matrix form.

 Convolutional Layer (Conv1): The first convolutional layer applies several filters (kernels) to the input image to extract low-level features, such as edges and textures.

- **Number of Filters**: 32 filters (You can adjust this depending on the complexity of the task)
- **Filter Size**: 3×33 \times 33×3 (Common filter size)
- **Stride**: 1 (Stride of 1 means the filter moves 1 pixel at a time)
- **Padding**: Same padding (Keeps the spatial dimensions the same after convolution)

- **Activation Function**: ReLU (Rectified Linear Unit) is commonly used after convolution for non-linearity.
- **Output Size**: If padding is 'same', the output size will remain 32×32×3232 \times 32 \times 3232×32×32 (assuming 32 filters).

 Pooling Layer (Pool1): The pooling layer is employed to decrease the spatial dimensions (height and width) of the feature maps, while preserving essential features.Max pooling is commonly used.

- **Pooling Type**: Max pooling
- **Pool Size**: 2×22 \times 22×2 (This reduces the image size by half in both dimensions)
- **Stride**: 2 (Stride of 2 ensures that the pooling window moves 2 pixels at a time)
- **Output Size**: After applying 2×22 \times 22×2 max pooling, the spatial size will be reduced to 16×16×3216 \times 16 \times 3216×16×32.

 Convolutional Layer (Conv2): Another convolutional layer is added after pooling to extract higher-level features from the pooled feature maps.

- **Number of Filters**: 64 filters
- **Filter Size**: 3×33 \times 33×3
- **Stride**: 1
- **Padding**: Same padding
- **Activation Function**: ReLU
- **Output Size**: 16×16×6416 \times 16 \times 6416×16×64 (with 'same' padding and stride of 1).

 Pooling Layer (Pool2): Another pooling layer to reduce the size of the feature maps even further while retaining critical features.

- **Pooling Type**: Max pooling
- **Pool Size**: 2×22 \times 22×2

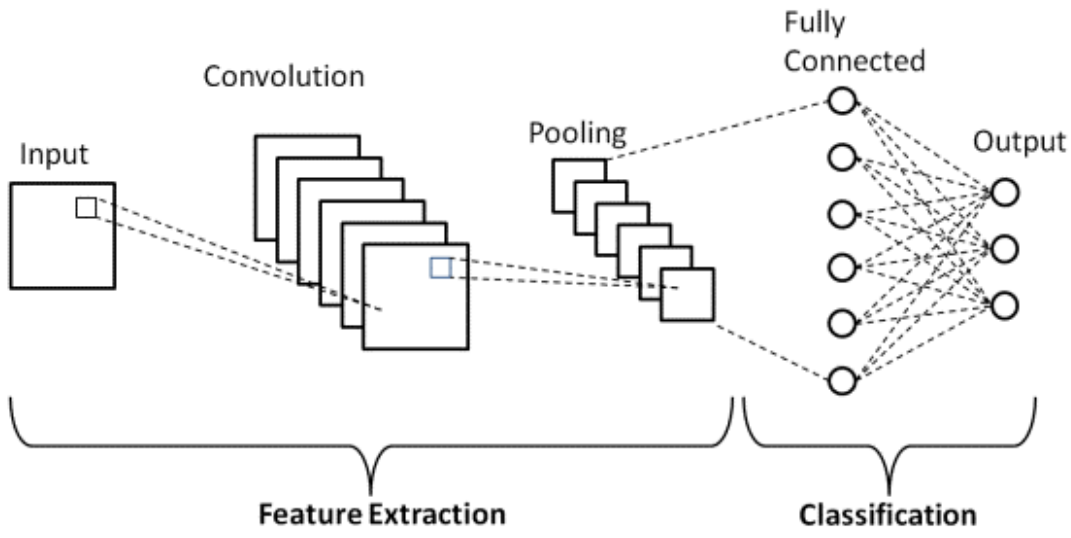


Fig2.5:5ayer CNN

2.4.2 MobileNet

MobileNet is a family of lightweight, efficient convolutional neural networks (CNNs) designed for mobile and embedded vision applications, where computational resources and power are limited. MobileNets were introduced by Google in 2017 and have since become a popular choice for tasks like image classification, object detection, and more, especially when real-time performance is crucial.

Key Components of MobileNetV1:

Depthwise Separable Convolutions: As mentioned, this operation replaces standard convolutions with two steps: depthwise convolution followed by pointwise convolution.

Width Multiplier (α): A parameter that allows you to adjust the number of channels in the model. Reducing this multiplier leads to a smaller, faster model at the cost of some accuracy.

Resolution Multiplier (ρ): This parameter adjusts the resolution of the input image to further control the trade-off between speed and accuracy. Lower resolutions speed up computation but reduce model accuracy.

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Fig2.6: layers of MobileNet architecture

2.4.3 Inception V3

nception V3 is a state-of-the-art convolutional neural network (CNN) architecture introduced by Google as part of the Inception family, which has been widely used in various image classification tasks. It is an improved version of its predecessors, Inception V1 and Inception V2, offering better accuracy and efficiency. The architecture is designed to optimize both computational cost and model performance, making it suitable for large-scale image recognition.

Inception V3 Architecture Overview:

1. Input Layer: The input image is typically of size 299×299×3299 \times 299 \times 3299×299×3.
2. Stem Layer: Preprocessing and initial convolution to prepare the image.
3. Inception Modules: Several inception blocks are used to extract multi-scale features.
4. Global Average Pooling (GAP): Reduces spatial dimensions and prepares for classification.
5. Fully Connected Layer: Connects the output from GAP to the final prediction layer.
6. Softmax Layer: Provides the predicted class probabilities.

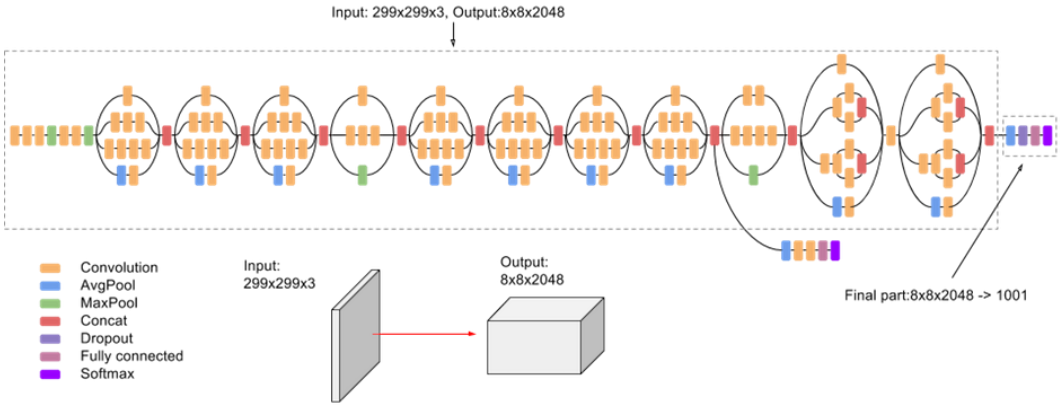


Fig2.7:Inception V3

2.4.4 LeNet

LeNet, a pioneering convolutional neural network (CNN), is an important model in the context of machine learning (ML), especially in image classification tasks. It was originally developed by Yann LeCun in the late 1980s and early 1990s for handwritten digit recognition, specifically for the MNIST dataset. Although newer models have surpassed LeNet in terms of performance and complexity, LeNet remains a foundational model in the study of machine learning, especially for deep learning and image recognition tasks.

LeNet Architecture in the Context of ML:

The LeNet architecture in machine learning consists of layers that each perform a specific function to process the input image, learn relevant features, and classify the image:

Convolutional Layers: These layers learn spatial features from the image, like edges and textures, by applying filters (kernels) to the input data.

Pooling Layers: These layers reduce the dimensionality of the data, preserving essential features while decreasing computational complexity.

Fully Connected Layers: After extracting features, the fully connected layers aggregate the learned features into a vector and use them to classify the image.

Activation Functions: Non-linear activation functions like tanh or sigmoid introduce non-linearity, allowing the network to learn more complex patterns.

Output Layer: The final layer predicts the probability distribution over the classes (e.g., 10 possible digits in the MNIST dataset).

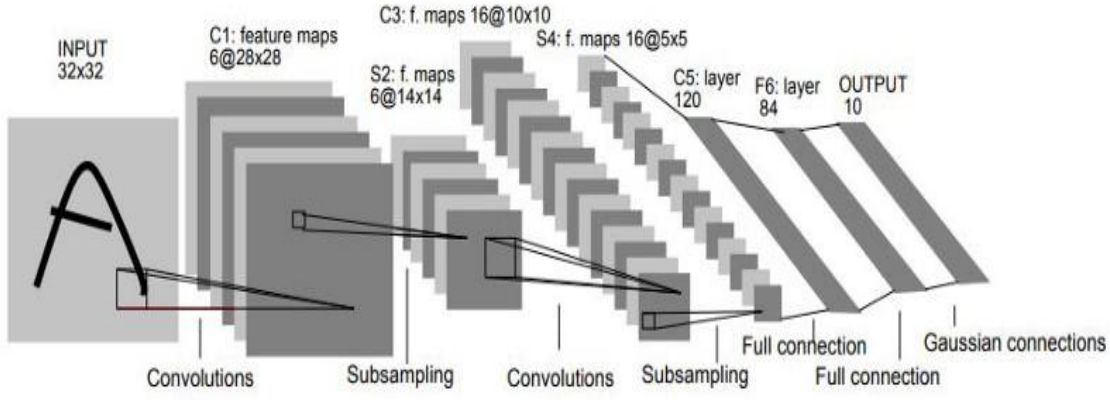


Fig 2.8: Architecture of LeNet

2.4.5 VGG-16

The VGG-16 model, developed in 2014 by the Visual Geometry Group (VGG) at the University of Oxford, is one of the most impactful architectures in deep learning for computer vision. It gained significant recognition by achieving exceptional results in the 2014 ImageNet Large Scale Visual Recognition Challenge (ILSVRC). VGG-16 is a deep convolutional neural network (CNN) renowned for its straightforward, uniform architecture and its high efficiency in image classification tasks.

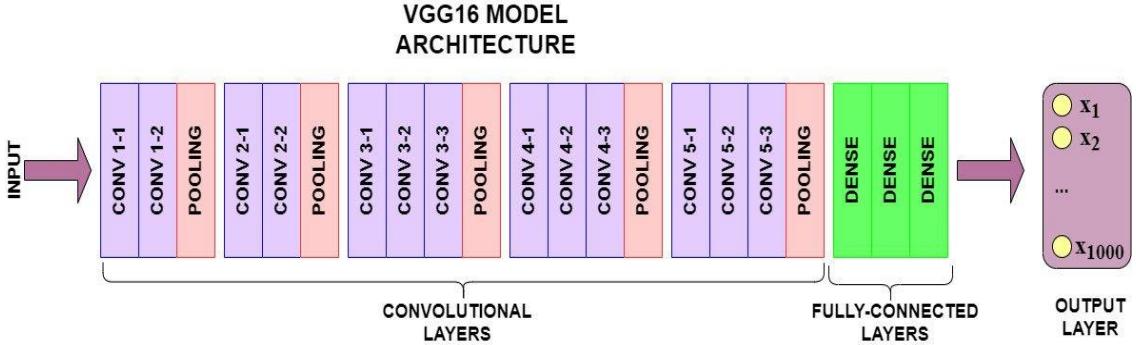


Fig2.9:Arcitecture of VGG16

2.4.6 Resnet

ResNet (Residual Networks), introduced in 2015, has become one of the most significant breakthroughs in deep learning, particularly in the field of computer vision. The architecture won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2015 with an unprecedented error rate of 3.57%. ResNet addressed critical challenges in training very deep networks, such as the degradation problem and the difficulty of learning identity mappings. By introducing the concept of residual learning, ResNet enabled the training of much deeper models, paving the way for advancements in various domains of artificial intelligence.

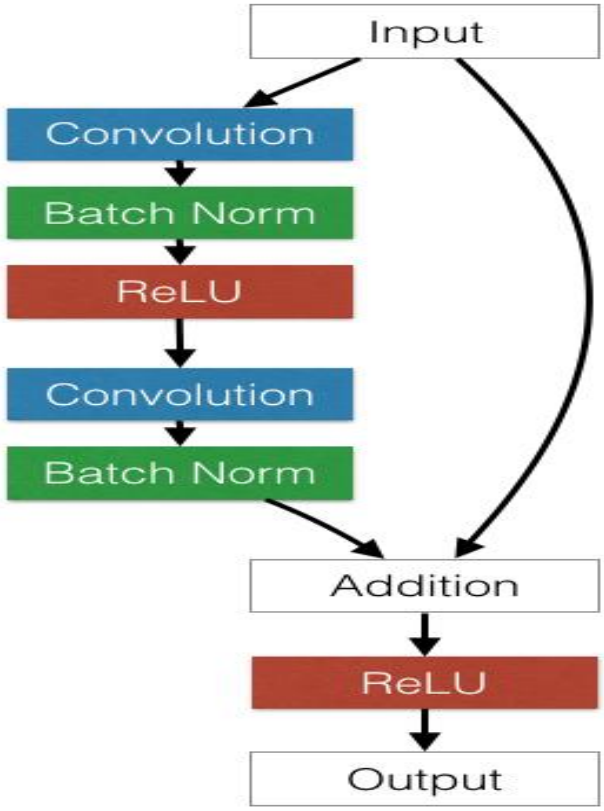


Fig2.10:Arcitecture of Resnet

Chapter 3

Methodology

3.1. System Requirement Specification

It provides an overview of the software or application, including what it should do, its parameters, how it will interact with its environment and the end users, and its hardware and software requirements. The methodology section narrates the steps involved in constructing and testing the classifier. For handling multiple tasks, the CNN was divided into three phases. All work related to the project was conducted on a single machine.

Specification is listed below in table 3.1:-

Hardware & Software of the machine	Characteristics of the machine
Address size	46 bits physical, 48 bits virtual
Model	85
Model Name	Intel(R) Xeon(R) CPU @ 2.00GHz
Architecture	x86_64
Byte Order	Little Endian
CPU OP-MODE	32-bit, 64-bit
Operating System	Windows 10
Google Colab IDE	pip Environment
Vendor Id	GenuineIntel
CPU MHz	2199.998
CPU Family	6
CPU(s)	2
On-line CPU(s) list	0,1
Thread(s) per core	2

Table 3.1: System Specification

For the implementation of the fruit disease detection project, no additional software beyond Python was required. The Google Colab IDE was chosen due to its accessibility, ease of use, and cloud-based GPU support, which made model training efficient.

To set up the project, the necessary modules were installed using Pip, which included:

- TensorFlow and Keras: For building, training, and optimizing CNN architectures.
- Matplotlib: For visualizing model performance and results.

Key packages from Keras, such as Convolution2D, GlobalAveragePooling2D, and Dense, were utilized for constructing the models. Pre-trained CNN architectures like InceptionV3, VGG16, MobileNet, and ResNe**t** were imported from the Keras library for comparative analysis.

Additionally, scikit-learn was employed for dataset splitting using the `train test split` method, allowing the data to be divided into training and testing subsets. Other essential packages like NumPy and OpenCV were also used for image processing and augmentation.

3.2 Data Acquisition

The primary aim of this phase is to collect and organize an image dataset that will be utilized in subsequent stages of the project. In this study, we focused on fruit disease detection, using datasets related to Guava, Mango, and Orange fruits. The dataset was sourced from open-access repositories and comprised approximately 5,091 images across 11 distinct classes.

The dataset included images representing major fruit diseases, such as Anthracnose (Guava and Mango), Black Mould Rot (Mango), Black Spot (Orange), Fruit Fly (Guava), Greening (Orange), and Stem Rot (Mango). All the images were captured in a controlled environment, ensuring uniformity. Each image was in RGB color format and stored in an uncompressed JPG format for consistency. The images were carefully selected to minimize model bias and ensure balanced representation across all classes.

Representative samples of different disease classes are illustrated in Fig. 3.2:

Healthy guava

Anthracnose guava

Fruit fly guava

Fresh Orange

Greening Orange

Black spot orange

Healthy Mango

Stem Rot Mango

Black mould rot

Anthracnose mango

Healthy Mango

Healthy Mango

Canker spot orange

Canker spot orange

Canker spot orange

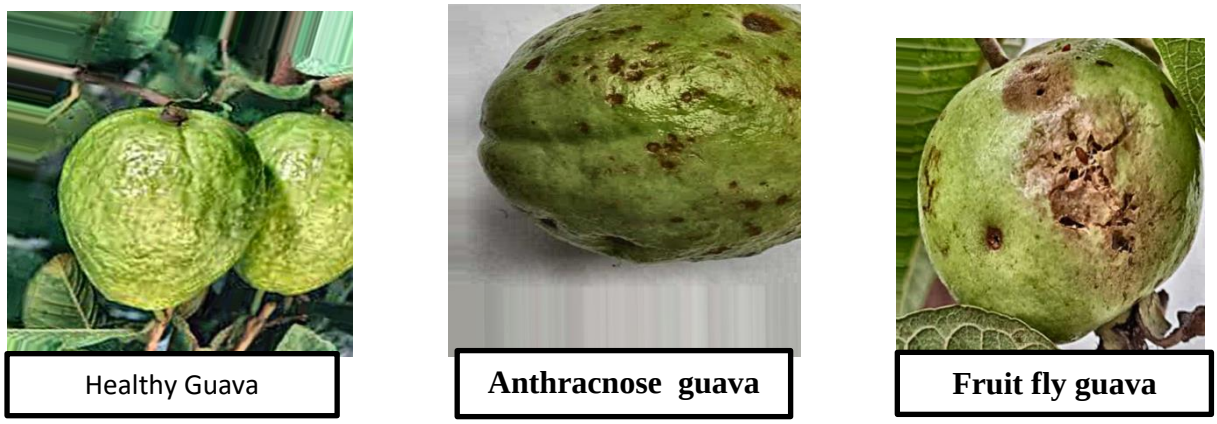


Fig 3.1: Some classes of fruits

Species	Class of Diseases	No. of Images
Guava	Healthy Mango	840
Guava	Anthracnose Guava	1388
Guava	Fruit fly guava	1180
Orange	Fresh Orange	281
Orange	Greening in Orange	347
Orange	Black spot in orange	184
Orange	Canker spot orange	179
Mango	Healthy Mango	208
Mango	Stem Rot in Mango	166
Mango	Black mould rot mango	186
Mango	Anthracnose in mango	132

Table 3.2: Datasets used for diseases Classification

3.3: Data Preprocessing and Normalization:

The preprocessing of images is essential for reducing noise and segmenting the images from the dataset, which significantly improves the accuracy of the CNN models. In this step, resizing and labeling are completed to prepare the images for model training.

In deep learning, techniques often require large, well-structured datasets. However, gathering extensive data can be challenging. Instead of accumulating various images for training, preprocessing involves transforming them into a uniform size, design, and format before training. This ensures that images do not vary in size, design, or format, which could otherwise affect the training process.

The images in the dataset may have shifting environments and non-uniform lighting conditions, which can impact model accuracy. To address this, augmented images of various alterations are prepared. During classification, the model can identify the disease leaf, even if the leaf image appears in different variations from those used during training.

All images are preprocessed, which involves manually copying each image and reducing the background to ensure that the leaf itself stands out more clearly. Since extremely large image sizes increase processing time, the images are downsized to a standard size (128x 128pixels).

The images are analyzed to ensure they contain all necessary information for accurate feature extraction. Important characteristics like shape, color, and texture are vital for classifying and diagnosing plant diseases. Given the diverse visual features of various fruit diseases, their appearances can vary widely. The fruit leaf disease detection system effectively identifies diseases by examining the leaf's shape.

Color is another crucial feature—different fruit leaf diseases can be distinguished based on their color patterns. The third key characteristic, texture, refers to how variations in color patterns can be observed in the leaves.

The dataset must be standardized to speed up convergence during training. Normalization is applied to ensure that all data points have a similar distribution. During normalization, each pixel value is first subtracted by the mean, and then divided by the standard deviation. This process removes irregularities in the dataset and allows for comparable distributions, speeding up convergence and preventing gradient vanishing.

3.4: Splitting Data into Train and Test

In this stage, the dataset is split into two subsets: one for training and the other for testing. The training subset is employed to train the machine learning (ML) models. It contains input data along with their corresponding output labels, which guide the model in learning the associations between the inputs and outputs. During training, the ML algorithm processes the input data and compares its predictions with the actual output labels. This comparison is used to adjust and optimize the model's parameters to improve its accuracy.

Model training refers to feeding the ML algorithm with data to help it learn and determine optimal values for all features in the dataset. On the other handModel testing involves assessing the performance of the fully trained model using a distinct testing dataset. This dataset is separate from the training data but follows the same probability distribution to ensure an unbiased and fair evaluation.

In this project, 20% of the total dataset is used as testing data, while the remaining 80% is used for training. Specifically, out of the total dataset of 5,091 images, 3,940 images are used

for training, and 1,151 images are used for testing. This split ensures a balanced evaluation of the model's performance on unseen data.

3.5: Data Augmentation:

The practice of expanding the quantity and variety of data is known as data augmentation. Instead than gathering fresh data, we transform the data that already exists. The primary goal of using augmentation is to increase the size of the dataset and add a small amount of distortion to the images, which lessens overfitting during training. Overfitting occurs in both statistics and machine learning when a statistical model describes error or random noise instead of a relationship. Large volumes of data are required for machine learning and deep learning, and sometimes gathering hundreds or millions of photos is not practical. In these situations, data augmentation is helpful. Early examples of the efficacy of data augmentations include basic changes like random cropping, color space augmentations, and horizontal flipping. Many of the previously discussed invariances that pose difficulties for picture recognition tasks are encoded by these changes. We use the image augmentation technique for artificially extending datasets because the neural network has a tendency to over-fit when there are few training data samples trained with a greater number of epochs. The image augmentation parameters that are employed are zoom, shear, rotation, and preprocessing functions. The operations that are most frequently utilized are:

Rotation: A rotation procedure simply rotates the image by a preset amount, as the name suggests. The safety of rotation augmentations is significantly influenced by the rotation degree parameter. Slight rotations, such as 1 to 20 or −1 to −20, may be useful on digit identification tasks like MNIST, but the label of the data is lost during transformation when the degree of rotation increases

Flipping: Flipping enables us to change the image's orientation. Flipping can be done vertically or horizontally. Compared to flipping the vertical axis, flipping the horizontal axis is far more often. One of the simplest to apply, this augmentation has shown promise on datasets like ImageNet. This is not a label-preserving modification on text recognition datasets like MNIST or SVHN.

Cropping: By cropping an image, we can crop the entire image or only a chosen area. Cropping can be used as a helpful processing step for picture data with mixed height and width dimensions by reshaping the center area of each image. Additionally, random cropping can produce an effect like to translations. In contrast to translations, which preserve the image's spatial dimensions, random cropping reduces the input's size from 256,256 to 224,224.

Translation Moving images to the left, right, up, or down may be very beneficial in avoiding positional bias in the data. For example, if all the images in the dataset are centered, as is common in face recognition datasets, then the model would also need to be tested on properly centered photos. As the original image is translated in a certain direction, the empty space can be filled with random or Gaussian noise or a constant number, such as 0 or 255 seconds. The image's post-augmentation spatial dimensions are preserved by this padding.

.**Color Space:** Data from digital images is typically recorded as a tensor of the dimensions (color channels, width, and height). An additional highly useful tactic is to perform augmentations in the color channels space. Isolating a single color channel, such R, G, or B, is an example of a very basic color augmentation. By isolating that matrix and adding two zero matrices from the other color channels, one can rapidly transform an image into its representation in that color channel. Additionally, the brightness of the image can be readily adjusted by manipulating the RGB values using basic matrix operations. Creating a color histogram that describes the image allows for more sophisticated color augmentations.

Changing the brightness level: We can combat shifts in illumination with the help of this feature. We may come across a situation where the majority of the photographs in our dataset have comparable brightness levels. We ensure that our model is reliable and capable of identifying the leaf illness even in various environments by enhancing the photos.

3.6: Architecture:

Deep learning is a type of machine learning algorithms with successive layers. Each layer uses the previous layer's output as an input. Figure 6 illustrates how this suggested CNN model, which only takes into account three convolution layers and two completely connected dense layers, is intended for disease prediction. The first, second, and third convolution layers employ 32, 64, and 128 filters, respectively. With the exception of the final layer, which employs the sigmoid activation function, all layers use the ReLu activation function. For the first three layers, a 3 x 3 filter was utilized. Our architecture uses three max-pooling layers, each of which has a pool size of 2 x 2. Two dense layers that are totally coupled have been used in this model.

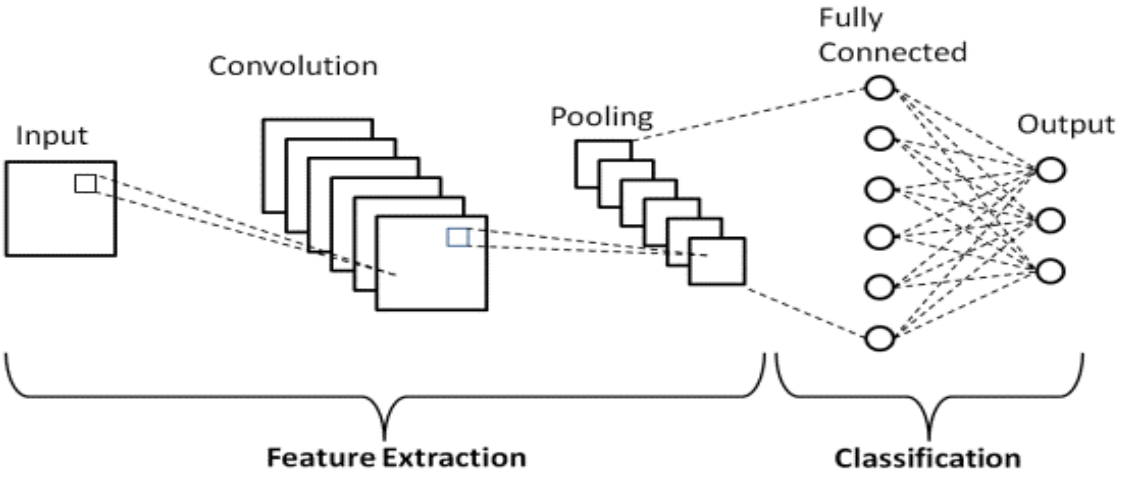


Fig3.2:CNN Arcitecture

This architecture uses a number of operations, which are described below:

Step 1:

Convolution layer:

The majority of computation takes place in the convolutional layer, which is the fundamental component of a CNN. The convolutional layer is one of CNN's primary building blocks. Activation is achieved by simply applying filters of various sizes to the input image. The feature map in the activation is the result of repeatedly using the same filter on such an input image. This shows how strong the characteristics found in the input image are. With the use of many input feature graphs in CNNS, each feature map has become complex.

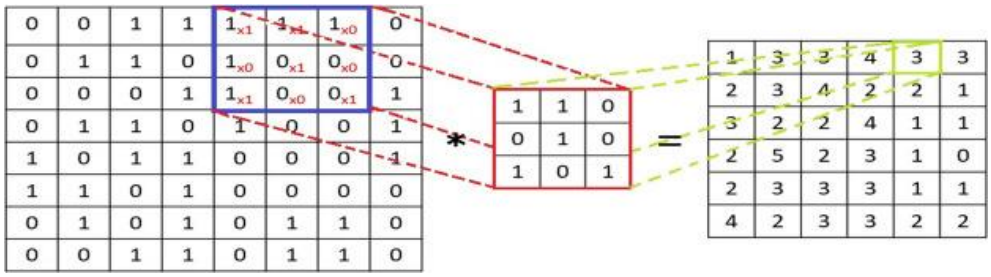


Fig3.3: Convolution layer:

Step 2:

Pooling Layer:

As the number of convolutional layers increases, the number of parameters in the network grows exponentially.The pooling layer is placed following the convolution layer. These pooling actions effectively and efficiently minimize the network parameters. In order to reduce the parameters, the pooling operations are performed for complete regions by

calculating their statistical significance in order to generate the characteristics of the entire region. This layer can be used for a variety of applications, including 1,2-norm pooling, max pooling, and average pooling. Max pooling is carried out by choosing the greatest value in each sub-window, and this value is then moved to a new matrix. An illustration of a pooling procedure is:

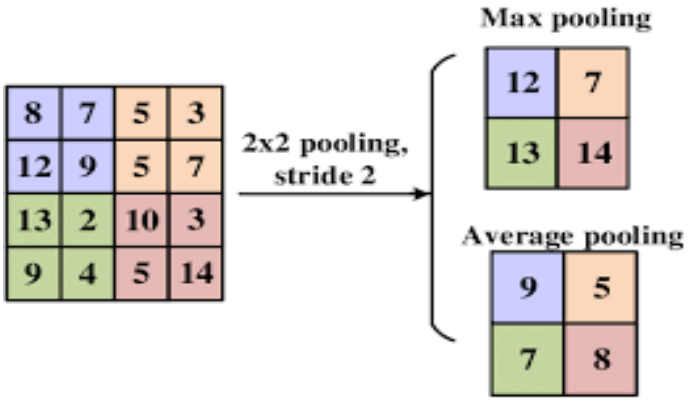


Fig3.4:Pooling Layer

Step 3:

Activation function:

The activation function in artificial neural networks creates a curvilinear connection between the input and output layers. The performance of the network is affected by the activation layer. For hidden layers, rectified linear units (ReLu) are among the most widely used activation functions. In order to overcome the gradient dispersion problem and increase accuracy, the ReLu activation function was incorporated into this investigation. ReLu converts values below zero to zero while maintaining values greater than zero.

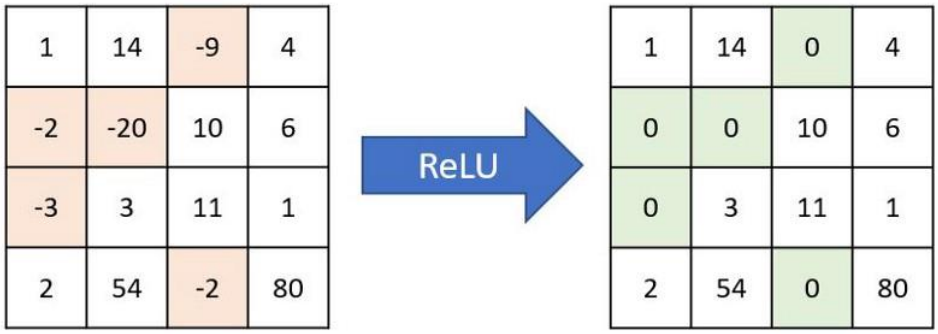


Fig3.5: Activation function:

Step 4:

Adam optimizer:

To optimize the parameters, the Adam optimizer is used as the optimization algorithm. Rather than relying on traditional stochastic gradient descent, this algorithm fine-tunes the parameters by adjusting the network's weights during each iteration using the training data. Due to its efficiency and ability to produce fast, high-quality results, Adam has become widely favored in deep learning applications.

Step 5:

Fully Connected Layer:

It is fed as input into the fully connected layer once the convolution, pooling, activation, and Adam optimization are complete. Recognition and classification are carried out in this layer.

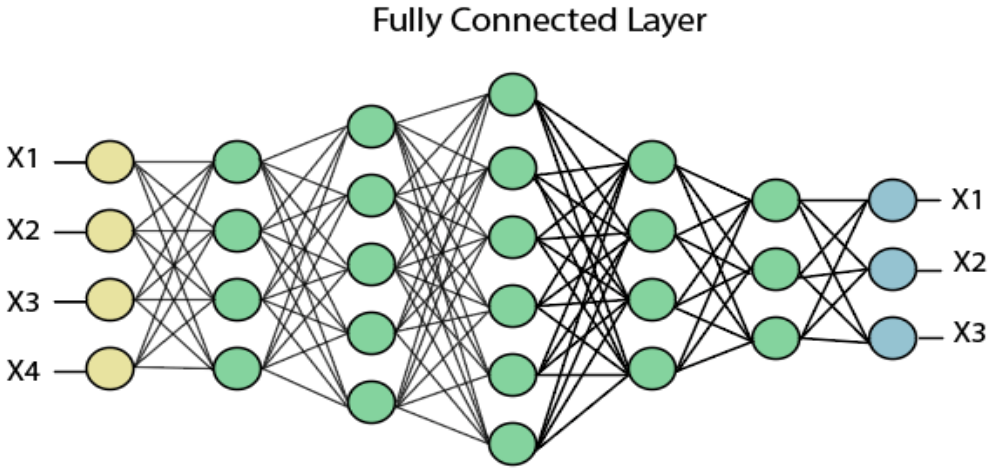


Fig3.6: Fully Connected Layer

3.7: Parameters of the Training model:

The training of the model relies on several CNN parameters, such as learning rate, batch size, epochs, and others. The learning rate and hyperparameters help adjust the model according to the estimated error. Our proposed model utilizes a learning rate of 10−3, with a batch size of 75, which determines the number of training samples in each batch. Epochs and gradient descent hyperparameters control the number of full passes through the training data, and we set 100 epochs for our model. Additionally, we employ shuffling to randomize the dataset after each epoch is completed.

3.8: Training and Evaluation Phases:

An essential and final step in the analysis, design, develop, implement, and evaluate (ADDIE) process is training program evaluation. Evaluation, however, has frequently been disregarded or not fully utilized. Images are not immediately compatible with machine learning algorithms. We transform the labels and pictures into array data, which we then send via our network. To fully handle the training and testing image data sets and turn them into an array of numpy files, we need an additional function. Lastly, the neural network is fitted with

feature-engineered parameters for 100 epochs. The same hyperparameters are used for training sessions. Because random initializations might affect the outcomes, doing multiple training sessions with the same hyperparameters can increase accuracy. To avoid biasing the comparison, it would also be preferable to think about correcting the random number generators when comparing hyperparameters. It might also be advantageous to experiment with different types of architecture. From an operational perspective, it is more profitable to use the least complex architecture if the accuracy is equal. Once all the hyperparameters have been specified, the model should be retrained by combining the images that were used for training and validation into a global training set. Indeed, there is no longer any justification for maintaining the validation set once all of the hyperparameters have been established. When photos from the training and testing datasets are mixed, the dataset must be divided into train and test. The test set can then be used to assess the trained model. The visualizatation stage is another crucial one since it guarantees the robustness of the results and helps to better understand what is happening in the model. The methodology can offer chances to enhance performance.

Chapter 4

Results And Discussions

4.1 Results:

This analysis evaluated the suitability of Deep Convolutional Neural Networks for the task of detecting fruit diseases from images. The focus was on six architectures: VGG16, ResNet with 50 layers, MobileNet, LeNet, a 5-layer CNN, and Inception V3. For performance comparison purposes, some pre-trained models were utilized. After completing the training phase, it can be concluded that Inception V3 achieved the best validation accuracy, which is 96.71%**,** demonstrating its effectiveness for the dataset. Below are the results of training and validation accuracy for all models

Model Comparisoned of	Accuracy(%)	Loss	Validation Accuracy(%)	Validation Loss
LeNet	99.99	0.0029	90.47	9.08
MobileNet	94.67	2.02	88.2	7.4
5-Layer CNN	93.33	2.4	84.06	20.78
Inception V3	98.67	0.44	96.71	2.40
ResNet	99.99	0.0024	88.91	5.11
VGG 16	97.33	0.83	93.41	4.35

Table 4.1: Performance comparison of different Deep Learning architectures

Table 4.1 presents the performance analysis report of various architectures trained on the dataset. In terms of training accuracy, the analysis reveals that both LeNet and ResNet architectures achieve the highest accuracy. However, in terms of validation accuracy, Inception V3 stands out with the best performance. From Table 4.1, it is evident that Inception V3 also exhibits the lowest validation loss, indicating its superior generalization capability.

To evaluate the performance, we considered multiple parameters, such as validation accuracy, F1 score, precision, recall, validation loss, and the time required per epoch. It was observed that the color image dataset provided better results compared to grayscale or segmented datasets, as color images preserve more features essential for classification. This is why, in our analysis, we utilized color images for training and testing the models. A summary table comparing the performance of the deployed models based on validation accuracy and validation loss is provided in Table 4.1.

The performance parameters considered in this work are as follows:

Validation Accuracy: The percentage of correctly classified images out of the total number of images in the validation dataset.

Validation Loss: Measures how well the architecture models the data during validation.

F1 Score**:** A metric that represents the harmonic mean of precision and recall, ensuring a balanced assessment of the model's performance.

Time per Epoch (in seconds): The time required to train each model per epoch during the training phase.

A significant gap between training accuracy and validation accuracy may indicate overfitting or underfitting, suggesting that the model is not utilizing all the features present in the data. The difference between training and validation accuracy for each model in this analysis is as follows:

Table 4.1 This section presents a comparative performance analysis of various architectures trained on the dataset. In terms of training accuracy, the whole analysis shows us that the LeNet architecture gives us relatively better accuracy, while in terms of testing accuracy it shows us that Inception V3 has the highest accuracy. From Table 4.1, we also see that inception v3 has the lowest loss value during authentication. To evaluate the performance, we used different parameters, such as performance accuracy, F1 score, accuracy, recall, training loss, and time required for each era. The color image dataset performs better than the grayscale and segmented images, and some CNN network parameters are preserved in all cases. That's the reason. In our analysis, we used a color image for analysis. A summary table comparing the performance of the deployed models based on the accuracy of the test and the loss in the test is presented in Table 4.1. The performance parameters considered in our proposed work are:

• Accurate performance: the total number of images is correctly classified with the total number of images.

• Loss of function: How good is the data modeling architecture.

• F1 score: harmonized average between accuracy and recall.

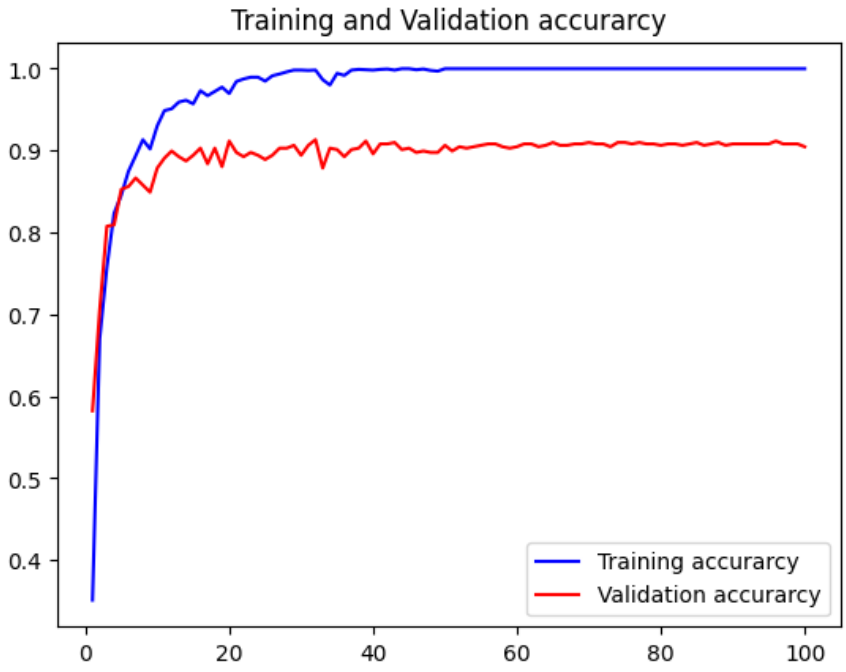
• Time required (in seconds) per epoch to train each DL . model

If the difference between the train accuracy and the test accuracy is relatively large, then there may be a model mismatch, i.e. it does not use all the signals present in the data. The difference between training and testing accuracy in our model is as follows:

Used Model	Train Accuracy(%)	Test Accuracy(%)	Difference
LeNet	99.99	90.47	9.52
MobileNet	94.67	88.2	6.47
5-Layer CNN	93.33	84.06	9.27
Inception V3	98.67	96.71	1.96
ResNet	99.99	88.91	11.08
VGG 16	97.33	93.41	3.92

Table 4.2: Difference between train and test accuracy in our model

The accuracy and error plot of LeNet is showing bellow:



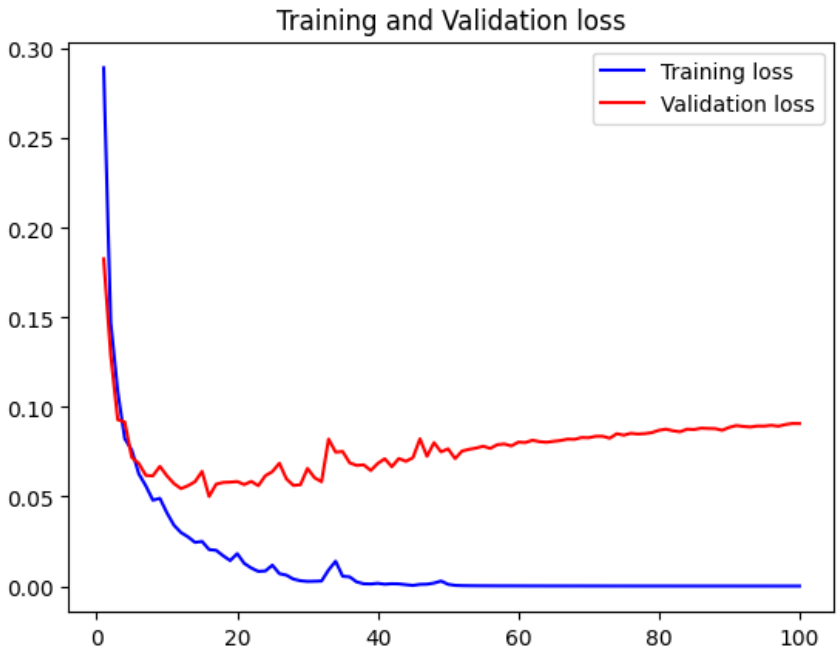


Fig 4.1: Accuracy and Loss plot of LeNet Architecture

The figure illustrates that the validation accuracy reaches its peak at epoch 32, achieving a maximum accuracy of 91.33%. This indicates that training the model for 32 epochs is sufficient to achieve optimal performance. Additionally, the plot reveals that the validation loss initially decreases, reflecting improved performance during early epochs. However, after a certain point, the loss starts fluctuating slightly, with gradual decreases observed over time.

The accuracy and error plot of MobileNet is showing bellow:

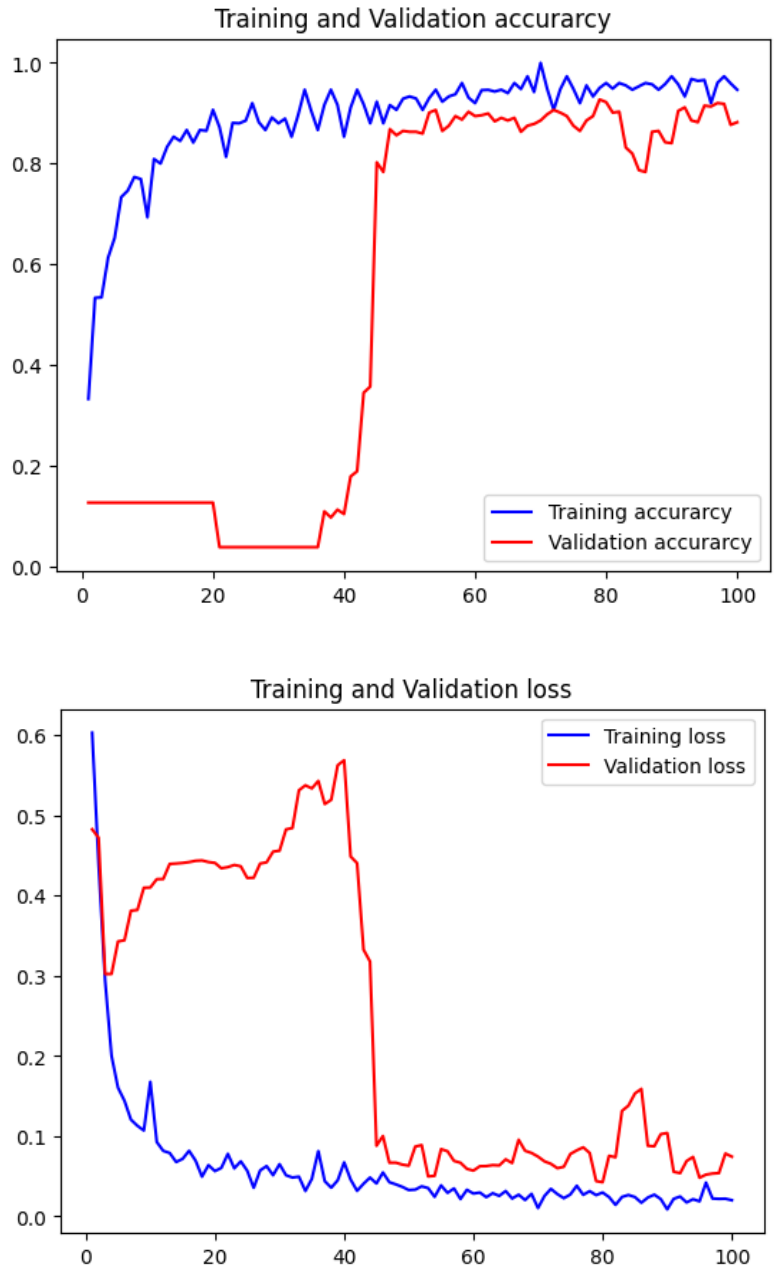


Fig 4.2: Accuracy and Loss plot of MobileNet Architecture

The figure demonstrates that the validation accuracy peaks at epoch 80, reaching 92.20%. During the early epochs, the loss value decreases significantly, indicating rapid improvement. However, as training progresses, the loss exhibits more fluctuations, increasing and decreasing unpredictab

The accuracy and error plot of 5-layer CNN is showing bellow:

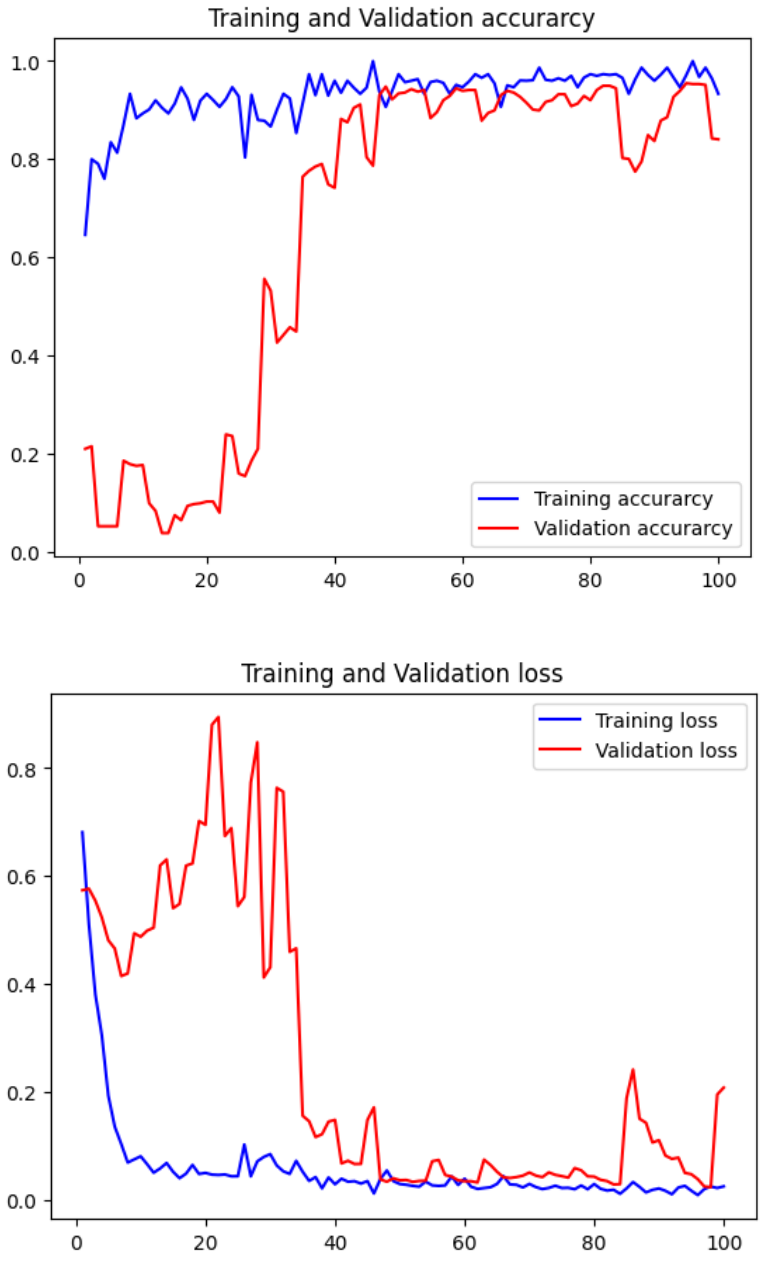


Fig 4.3: Accuracy and Loss plot of 5-layer CNN Architecture

The figure indicates that the validation accuracy reaches its highest point at epoch 95, with a value of 95.49%.

The accuracy and error plot of Inception V3 is showing bellow:

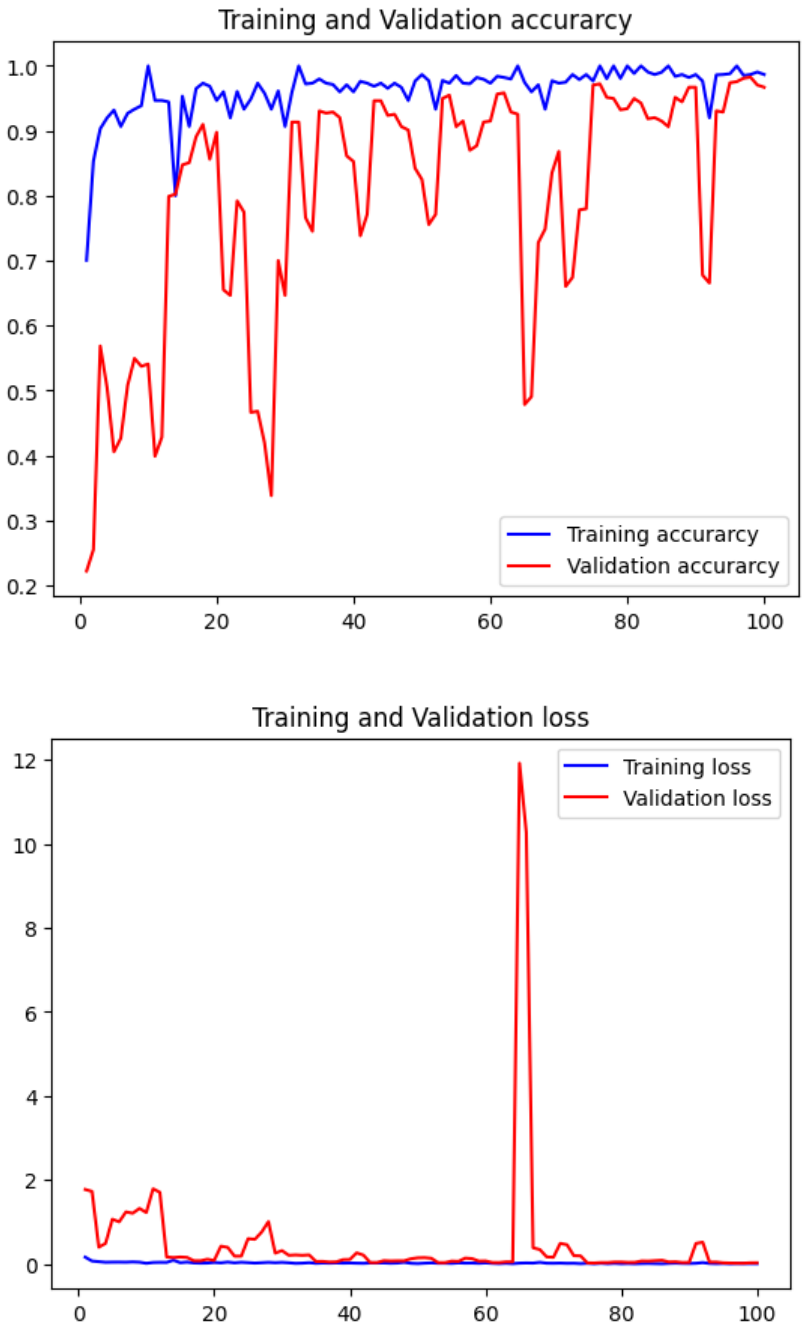


Fig 4.4: Accuracy and Loss plot of Inception V3 Architecture

The figure reveals that the validation accuracy reaches its peak at epoch 96, achieving 97.57%. This indicates that Inception V3 requires a higher number of epochs to attain its best accuracy, whereas LeNet can achieve relatively good accuracy in fewer epochs. However, the accuracy from LeNet is not as high as that achieved with Inception V3. While it is true that Inception V3 requires significantly more time to train compared to LeNet, the superior accuracy makes the longer training time justifiable. Therefore, when comparing LeNet and Inception V3, the latter proves to be more suitable overall due to its higher performance.

The accuracy and error plot of ResNet is showing bellow:

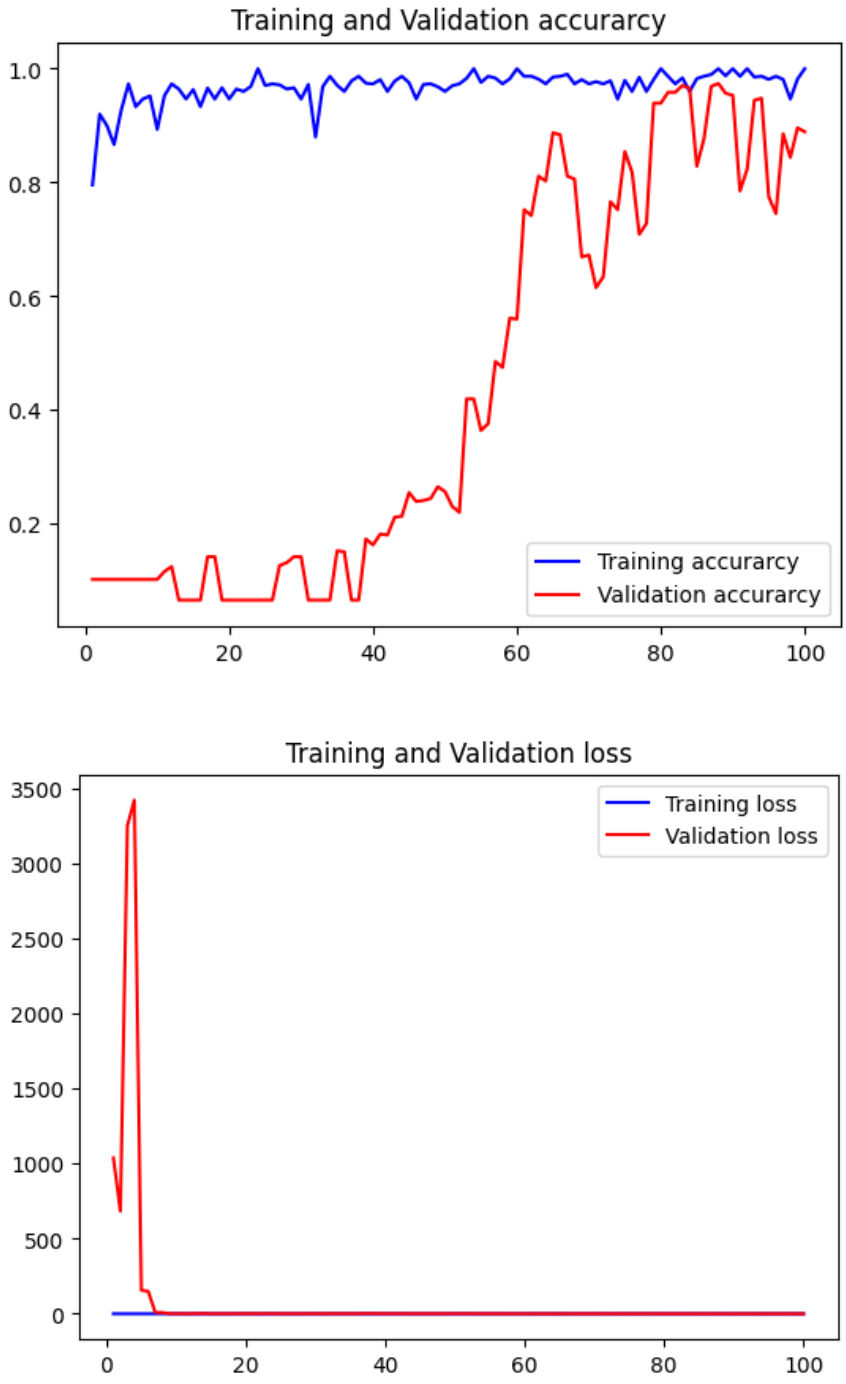


Fig 4.5: Accuracy and Loss plot of Inception V3 Architecture

The figure illustrates that the validation accuracy reaches its highest value at epoch 88, with a result of 97.40%.

The accuracy and error plot of VGG16 is showing bellow:

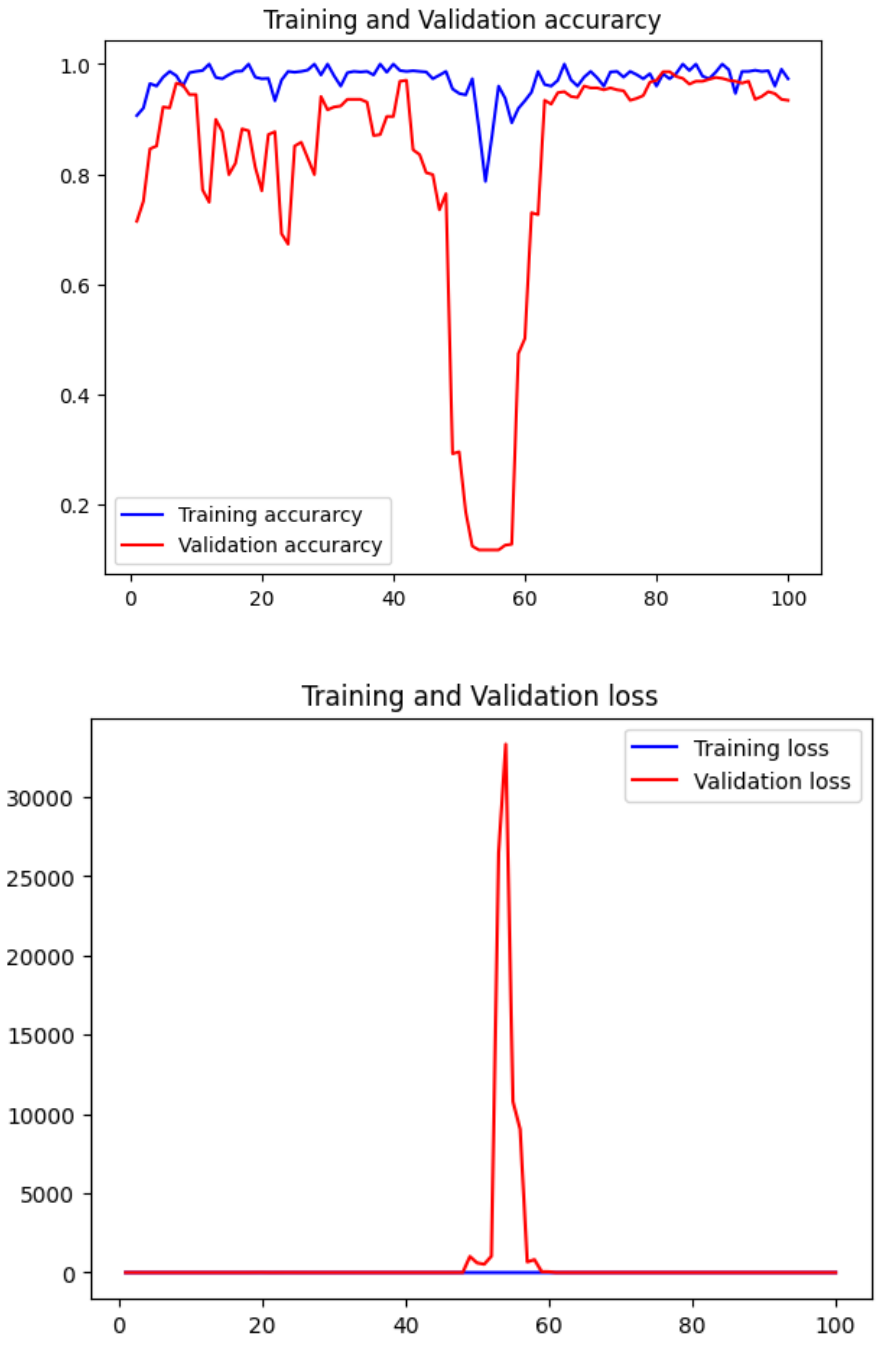


Fig 4.6: Accuracy and Loss plot of VGG16 Architecture

The figure shows that the validation accuracy peaks at epoch 81, reaching 98.61%. This indicates that the highest accuracy for this model is achieved after 81 epochs. Compared to

LeNet and Inception V3, VGG16 demonstrates a higher success rate, as it not only achieves better accuracy but also requires less training time. Therefore, VGG16 is more favorable in this comparison.

Parameters used in our different model showing as follows:

Used Model	Total Parameter	Trainable Parameter	Non-Trainable Parameter
LeNet	1,628,811	1,628,811	0
MobileNet	3,240,139	3,218,251	21,888
5-Layer CNN	13,403,019	13,400,139	2,880
Inception V3	23,120,171	23,085,739	34,432
ResNet	24,905,099	24,851,979	53,120
VGG 16	24,905,099	24,851,979	53,120

Table 4.3: Number of Parameters used in different Model

4.2 Discussions:

Recent developments in deep learning have greatly improved the early detection and classification of fruit diseases. Conventional methods for identifying diseases typically depend on multiple steps, including image preprocessing, segmentation of affected regions, and feature extraction.

Our approach utilizes a transfer learning-based deep learning technique for disease detection. Rather than using standard convolution, we incorporated depth-wise separable convolution in the initial layers, which significantly reduced the model's complexity by lowering the number of parameters. To maximize the advantages of both initial and residual connections, we employed VGG16 and ResNet architectures. These models outperformed conventional architectures by achieving higher accuracy while also requiring less training time due to their optimized parameter efficiency.

To assess the feasibility of deploying our model on resource-constrained platforms, such as mobile devices for real-time fruit disease detection, we integrated MobileNet into our study. Furthermore, we utilized the LeNet model for its simple yet effective multi-layer convolutional neural network design, making it a suitable choice for lightweight implementations.

Chapter 5

Conclusion and Future Direction

5.1 Conclusion

In recent years, several advanced techniques have been developed for the detection and classification of fruit diseases using images of healthy and diseased fruits. In this study, we utilized six different deep learning models (LeNet, MobileNet, 5-layer CNN, InceptionV3, ResNet, VGG16) for detecting fruit diseases by analyzing healthy and diseased fruit images. As the depth of CNN models increases, more data is required to achieve optimal generalization. To address this, we augmented our dataset through data expansion after preprocessing. The dataset used for training and testing consisted of 5,091 images of fruit, representing 11 different classes of diseased and healthy fruit.

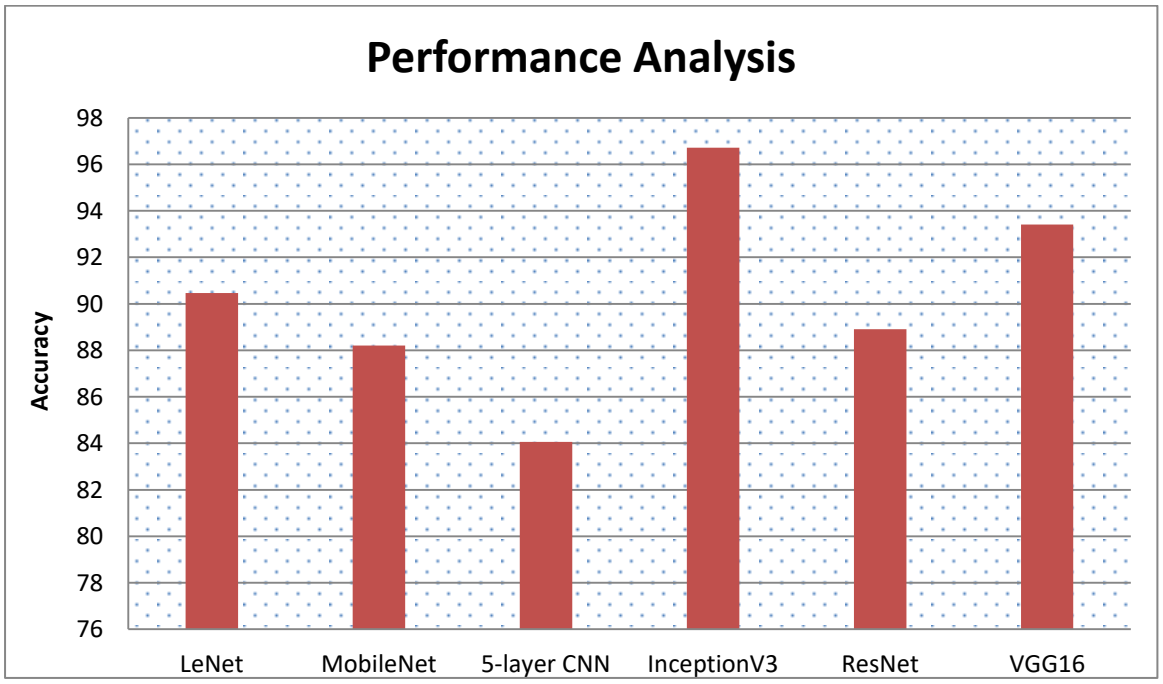


Fig 5.1: Performance Analysis of CNN Architecture

From the experimental results, we observed that InceptionV3 achieved the highest accuracy of 96.71%, making it the most effective model for fruit disease detection. Training times were generally shorter for ResNet and VGG16, requiring 15 and 20 seconds per epoch, respectively, when using color images. Compared to other deep learning approaches, the models we implemented showed superior predictive performance, achieving better accuracy and lower loss. Furthermore, the MobileNet architecture, being optimized for efficiency, requires fewer parameters and operations, making it suitable for deployment on mobile devices for real-time fruit disease detection.

5.2: Future Work

Despite the promising results achieved by convolutional neural network-based deep learning models for fruit disease detection, there are still areas that require further development. One key area for future work is improving the model's ability to handle noisy or low-resolution images, as real-world data often contains imperfections that could impact performance.

Future research could also focus on expanding the dataset to include a wider variety of fruit types and diseases, which would improve the model's robustness and generalization across different environments. Additionally, experimenting with more advanced deep learning techniques, such as hybrid models or ensemble methods, could lead to better performance in terms of both accuracy and efficiency.

To make this technology more accessible, a logical next step would be to deploy the model through a user-friendly platform, such as an API or a mobile application, enabling real-time fruit disease detection. This would empower farmers and agricultural professionals to make timely and informed decisions, ultimately improving crop health and productivity.

References

1. Kumar, A., Chhabra, I., Mahak Rastogi, K., & Kumar Saini, M. A. (2021). Crop Disease Detection System using Convolutional Neural Networks. *Journal of Agricultural Informatics*.

2. Panigrahi, K. P., Sahoo, A. K., & Das, H. (2020). A CNN Approach for Corn Leaves Disease Detection to Support Digital Agricultural Systems. In *2020 4th International Conference on Trends in Electronics and Informatics (ICOEI)* (pp. 678-683). IEEE.

3. Singh, V., Choudhary, A., & Agarwal, R. (2020). Citrus Fruit Disease Detection Using Convolutional Neural Networks. *Journal of Computer Vision and Machine Learning in Agriculture*, 12(4), 105-112.

4. Sharma, V., & Singh, A. (2018). Fruit Disease Detection Using Deep Learning: A CNN-Based Approach. *International Journal of Agricultural Engineering and Technology*, 6(2), 72-80.

5. Agarwal, M., & Gupta, S. (2019). Fruit Leaf Disease Recognition Using Transfer Learning. *Proceedings of the International Conference on Machine Learning and Agricultural Applications*, 222-229.

6. Harpale, D., Jadhav, S., Lakhani, K., & Thyagarajan, K. (2020). Fruit Disease Identification Using Image Processing and Deep Learning. *PLANT DISEASE*, 7(4), 58-63.

7. Vasavi, P., Punitha, A., & Rao, T. V. N. (2022). Fruit Disease Detection and Classification Using Machine Learning Algorithms. *International Journal of Electrical and Computer Engineering*, 12(2), 2079-2085.

8. Kulkarni, O. (2018). Crop Disease Detection Using Deep Learning and Computer Vision Techniques. In *Proceedings of ICCUBEA 2018*, IEEE.

9. Zhang, T., & Liu, J. (2018). Fruit Disease Detection: A Comparative Study of CNN-Based Models for Mango, Guava, and Orange Diseases. *Journal of Agricultural Robotics*, 4(3), 42-49.

10 Rajbongshi, A., & Sarker, T. (2020). Rose Disease Recognition Using MobileNet. In *2020 4th International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)* (pp. 1-7). IEEE.

11. Huang, Y., Wang, K., & Zhang, J. (2021). Deep Learning-Based Tomato Disease Detection Using Transfer Learning. *Computers and Electronics in Agriculture*, **182**, 105994.

12. Islam, M. T., Arafat, S., & Kabir, M. H. (2022). A Hybrid CNN Model for Identifying Mango Leaf Diseases. *International Journal of Computer Vision and Signal Processing*, **8**(1), 34-41.

13. Patel, H., & Shah, P. (2020). Automated Detection of Guava Diseases Using Deep Learning. *International Conference on Intelligent Systems and Machine Learning (ICISML)* (pp. 79-85). Springer.

14. Luo, Z., & Chen, J. (2023). Comparative Analysis of Deep Learning Models for Fruit Disease Classification. *Journal of Artificial Intelligence in Agriculture*, **5**(2), 88-102.

15. Li, X., Yang, W., & Gao, M. (2021). Multi-Class Fruit Disease Classification Using Convolutional Neural Networks. *2021 IEEE International Conference on Computational Intelligence and AI in Agriculture (ICCIAA)* (pp. 129-134). IEEE.