A Project Report on

CROP DISEASE DETECTION USING RESNET-50

Submitted in partial fulfillment of the requirement for the award of the Degree in

Bachelor of Technology

in

Computer Science and Engineering

Submitted By

BANDI SNEHA SRI LALITHA	**21031A0507**
HARI PRASAD NAIK AMGOTHU	**21031A0523**
CHINTHAPALLI VISWA	**21031A0512**
TRIPURA N V V S PRATYUSHA	**21031A0556**
ANTHERVEDIPALEPU BHANU PRAKASH	**21031A0504**

Under the esteemed guidance of

Dr. G. Madhavi

ASSISTANT PROFESSOR & HOD CSE (i/c)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UNIVERSITY COLLEGE OF ENGINEERING NARASARAOPET

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY KAKINADA

NARASARAOPET – 522601, PALNADU, ANDHRA PRADESH

2024-2025

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UNIVERSITY COLLEGE OF ENGINEERING NARASARAOPET

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY KAKINADA

NARASARAOPET – 522601, PALNADU, ANDHRA PRADESH

2024-2025

CERTIFICATE

This is to certify that the dissertation entitled "**CROP DISEASE DETECTION USING RESNET-50**" that is being submitted by **BANDI SNEHA SRI LALITHA** (21031A0507), **HARI PRASAD NAIK AMGOTHU** (21031A0523), **CHINTHAPALLI VISWA** (21031A0512), **T N V V S PRATYUSHA** (21031A0556), **ANTHERVEDIPALEPU BHANU PRAKASH** (21031A0504), in partial fulfilment for the award of Bachelor of Technology in Computer Science and Engineering to the University College of Engineering Narasaraopet, Jawaharlal Nehru Technological Kakinada is a record of bonafide work carried out by them under my guidance and supervision.

The results embedded in this dissertation have not been submitted to any other university/institute for the award of any degree/diploma.

PROJECT SUPERVISOR
Dr. G. Madhavi
Assistant Professor & HOD (i/c)
Department of CSE
UCEN JNTUK

HEAD OF THE DEPARTMENT
Dr. G. Madhavi
Assistant Professor & HOD (i/c)
Department of CSE
UCEN JNTUK

EXTERNAL EXAMINER

DECLARATION

We hereby declare that the work described in the project report entitled "**CROP DISEASE DETECTION USING RESNET-50**" which is submitted by us in partial fulfilment of the requirements for the award of the degree Bachelor of Technology in the department of **COMPUTER SCIENCE AND ENGINEERING** to the college, University College of Engineering Narasaraopet, Jawaharlal Nehru Technological University Kakinada, Andhra Pradesh is a record of original and independent research work done by us during the academic year 2024-2025 under the supervision of **Dr. G. Madhavi**, Assistant Professor & HOD (i/c) of CSE Department. The work is original and has not been submitted for the award of any Degree or Diploma or Associateship or Fellowship or other similar title to this or any other university.

Name of the Student	**Roll No**	**Signature**
BANDI SNEHA SRI LALITHA	**21031A0507**	
HARI PRASAD NAIK AMGOTHU	**21031A0523**	
CHINTHAPALLI VISWA	**21031A0512**	
TRIPURA N V V S PRATYUSHA	**21031A0556**	
ANTHERVEDIPALEPU BHANU PRAKASH	**21031A0504**	

PLACE :

DATE :

ACKNOWLEDGEMENT

It is needed with a great sense of pleasure and immense sense of gratitude that we acknowledge the help of these individuals. We express our sincere thanks and gratitude to many people who helped, supported, encouraged, and challenged us throughout our project and academic program. We take privilege to express our heartfelt gratitude to Project Supervisor and Head of the Department **Dr. G. MADHAVI**, Assistant Professor & HOD (i/c) and our Project Coordinator **Mrs. B. JYOTHI**, Assistant Professor (c) of CSE Department for their valuable suggestions and constant motivation that greatly helped us in successful completion of the project.

We express our sincere thanks to **Dr. Y. S. KISHORE BABU**, Vice-Principal(i/c), UCEN JNTUK and **Prof. CH. SRINIVASA RAO**, Principal, UCEN JNTUK for their support and constant motivation in successful completion of the project. We are thankful to all the faculty members for extending their kind cooperation and assistance. Finally, we are extremely thankful to our parents and friends for their constant help and moral support.

TEAM MEMBERS

BANDI SNEHA SRI LALITHA	21031A0507
HARI PRASAD NAIK AMGOTHU	21031A0523
CHINTHAPALLI VISWA	21031A0512
TRIPURA N V V S PRATYUSHA	21031A0556
ANTHERVEDIPALEPU BHANU PRAKASH	21031A0504

ABSTRACT

Agriculture plays a vital role in food security and economic stability, but crop diseases significantly impact yields and farmer incomes. Traditional disease identification methods are time-consuming and prone to human error, necessitating automated solutions using deep learning. Existing systems primarily rely on VGG16 for crop disease detection, achieving high accuracy but suffering from high computational costs and inefficiencies, making real-time deployment on resource-constrained devices challenging. Our proposed system, **Crop Disease Detection Using ResNet-50,** enhances detection accuracy while optimizing performance by leveraging residual learning to ensure efficient gradient flow and reduced computational overhead. Our model achieves an accuracy of 99.04%, whereas the existing system predicts diseases for only two crops—grapes and tomatoes. In contrast, our approach extends detection capabilities to 38 different crop diseases. Additionally, our system integrates **Planty**, an AI-driven chatbot that provides farmers with real-time disease insights, treatment recommendations, and interactive support. By improving accuracy, efficiency, and accessibility, our approach makes crop disease detection more scalable and practical for real-world agricultural applications.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABELS

LIST OF ABBREVIATIONS

S.NO	ABBREVIATION	EXPANSION
1	CNN	Convolutional Neural Networks
2	DL	Deep Learning
3	SRS	Software Requirements specification
4	API	Application Program Interface
5	UML	Unified Modelling Language
6	AI	Artificial Intelligence
7	NLP	Natural Language Processsing
8	LLM	Large Language Model
9	VGG	Visual Geometry Group

Chapter 1
INTRODUCTION

1.1 INTRODUCTION

In recent years, advancements in artificial intelligence (AI) and deep learning have significantly improved the accuracy and efficiency of crop disease detection systems. These models leverage state-of-the-art computer vision techniques and machine learning algorithms to identify plant diseases at an early stage, helping farmers take timely preventive measures. Given the critical role of agriculture in global food security, developing robust and scalable disease detection models is essential to minimize crop losses and enhance productivity.

Deep Learning for Crop Disease Detection:

Deep learning models, particularly Convolutional Neural Networks (CNNs), have demonstrated remarkable performance in analyzing plant images for disease classification. Trained on large datasets of infected and healthy crop images, these models learn intricate patterns and features that distinguish various plant diseases with high accuracy.

Challenges in Crop Disease Detection:

Detecting crop diseases poses several challenges due to variations in environmental conditions, plant species, and disease symptoms. Factors such as lighting, occlusions, and similarities between different diseases make the classification task complex. Addressing these challenges requires robust preprocessing techniques and diverse training datasets.

Model Architecture:

A typical crop disease detection model follows a multi-stage process:

- **Image Acquisition**: High-resolution images of crops are captured using smartphones, drones, or IoT-based agricultural sensors.
- **Preprocessing & Augmentation**: Images undergo normalization, resizing, and augmentation techniques to enhance model generalization.
- **Feature Extraction & Classification**: CNN-based models analyze the images, extracting relevant features and classifying them into predefined disease categories
- **Prediction & Recommendation**: The system provides real-time disease predictions along with possible treatments or preventive measures based on expert knowledge.

By integrating AI-driven crop disease detection systems into modern agricultural practices, farmers can reduce losses, improve yield quality, and ensure sustainable farming methods.

1.2 Significance of the project:

AI-powered crop disease detection helps farmers quickly identify plant diseases using deep learning and image processing techniques. By analyzing crop images, the system accurately classifies diseases, reducing the need for manual inspections. Early detection allows farmers to take timely action, minimizing crop losses and improving yield quality. The model also optimizes pesticide use, preventing unnecessary chemical application. Overall, it enhances agricultural efficiency and promotes sustainable farming practices.

1.3 Advantages and Disadvantages of CNN Based Crop Disease Detection Using ResNet - 50

System CNN-based crop disease prediction systems offer high accuracy in disease identification by recognizing intricate patterns in plant images. This enables early detection, preventing large-scale crop damage. These systems also automate diagnosis, reducing reliance on agricultural experts and making disease identification more accessible for farmers. Additionally, CNN models are adaptable to various crops and scalable, allowing large-scale disease monitoring through drones or satellite imaging.

However, these systems have some limitations. They require large and diverse datasets, and without enough high-quality images, predictions may be inaccurate. CNN models also demand high computational power, making them less accessible for small-scale farmers. Moreover, image quality and environmental factors like lighting and background noise can affect accuracy. Another drawback is their black-box nature, providing predictions without clear explanations. Additionally, most models rely on cloud-based processing, limiting offline functionality in remote agricultural areas.

Addressing these challenges through better datasets, lightweight models, and improved interfaces can make CNN-based crop disease prediction systems more reliable and widely usable.

1.4 Summary

In this chapter, we discussed the development of the CROP DISEASE DETECTION USING RESNET - 50 using deep learning and computer vision techniques. We provided an overview of how the system analyzes crop images to detect and classify diseases with high accuracy. The model helps farmers take timely action, reducing crop losses and improving yield quality. By leveraging AI, it minimizes manual inspections and optimizes pesticide use. Additionally, we outlined the key components involved in building this system to ensure its effectiveness in real-world agricultural applications.

.

Chapter 2

CROP DISEASE DETECTION MODEL Using ResNet - 50: A REVIEW

2. A REVIEW

2.1 A REVIEW ON CNN BASED CROP DISEASE DETECTION USING RESNET-50:

[1] Krizhevsky and Mohanty The AlexNet model, introduced by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton in 2012, was one of the first deep CNN architectures used in image classification. Inspired by its success, Mohanty et al. (2016) applied AlexNet to the PlantVillage dataset, achieving an accuracy of 99.35% in identifying 38 different plant diseases. Despite its effectiveness, AlexNet suffered from high computational costs and limited feature extraction capability due to its relatively shallow depth compared to later models.

[2] Simonyan & Zisserman and Sladojevic In 2014, Karen Simonyan and Andrew Zisserman developed VGG16 and VGG19, which improved upon AlexNet by using deeper networks with smaller convolutional filters. Sladojevic et al. (2016) applied VGG16 for leaf disease classification, reporting high accuracy in recognizing different plant diseases. However, the model required a large number of parameters, making training computationally expensive and prone to overfitting, especially when applied to small agricultural datasets.

[3] Szegedy & Too To improve computational efficiency while extracting meaningful features, Christian Szegedy et al. (2015) introduced InceptionNet (GoogLeNet). This architecture used parallel convolutional layers to capture multi-scale features. Later, Too et al. (2018) used Inception-v3 for classifying 14 plant diseases, achieving an accuracy of 96.3%. Despite its improvements over previous models, InceptionNet's performance was sensitive to background variations in images, requiring extensive preprocessing for accurate predictions.

[4] Howard & Fuentes With the increasing demand for mobile-friendly AI models, Andrew Howard et al. (2017) introduced MobileNet, which used depthwise separable convolutions to reduce computational complexity. Fuentes et al. (2019) applied MobileNet for real-time crop disease detection on mobile devices, demonstrating its potential for on-field applications. However, due to its reduced model depth, MobileNet had limitations in extracting complex patterns compared to heavier models like VGG16 and InceptionNet.

[5] Ferentinos Before the adoption of ResNet-50, researchers explored transfer learning, where pre-trained CNN models were fine-tuned for plant disease datasets. Ferentinos (2018) applied multiple pre-trained CNN architectures, including AlexNet, VGG16, and Inception-v3, to classify crop diseases. His study revealed that transfer learning significantly reduced training time while improving classification accuracy. However, deeper models without residual learning mechanisms still faced vanishing gradient problems, limiting their ability to scale effectively.

2.2 ISSUES IN CROP DISEASE DETECTION SYSTEM

Limited Feature Extraction in Traditional Models:

Early crop disease prediction models relied on traditional machine learning techniques such as Support Vector Machines (SVM) and Random Forest. These models required manual feature extraction, which limited their ability to recognize complex disease patterns. As a result, they often struggled with accuracy when applied to diverse datasets, making them less effective in real-world scenarios.

Insufficient Dataset Quality and Availability:

Many previous models were trained on small, region-specific datasets, leading to poor generalization when applied to different crops or environmental conditions. The lack of well-annotated and diverse datasets resulted in inconsistent model performance, reducing their reliability for large-scale agricultural applications.

Difficulty in Distinguishing Similar Diseases:

Previous models often faced challenges in differentiating between diseases with similar visual symptoms. Since handcrafted feature extraction methods were limited, they struggled to recognize subtle variations between diseases, leading to misclassification. This made them unreliable for farmers who needed precise disease identification for effective treatment.

2.3 MOTIVATION

Crop diseases significantly impact agricultural productivity, leading to severe economic losses for farmers. Traditional methods of disease detection require expert knowledge and manual intervention, which are time-consuming and often inaccessible to small-scale farmers. To address these challenges, our project leverages deep learning with ResNet-50 to automate disease identification with high accuracy. Additionally, a chatbot powered by the Gemini API assists farmers by providing disease-related information and potential remedies. This system bridges the gap between farmers and technological advancements, ensuring timely disease detection and actionable insights, ultimately improving crop health and yield.

2.4 PROBLEM STATEMENT

Crop diseases cause significant yield losses, and traditional detection methods are slow and require expert knowledge. This project aims to develop an AI-powered Crop Disease Detection Model using deep learning and computer vision to identify diseases from leaf images accurately. By leveraging CNNs and transfer learning, the system enables early detection, optimized pesticide use, and improved crop health, supporting sustainable farming.

2.5 SUMMARY

In this chapter, we discussed various research studies on crop disease detection models using deep learning and computer vision. We explored the technologies employed, including Convolutional Neural Networks (CNNs) and transfer learning, along with their advantages and limitations. While several models exist for detecting crop diseases, each has its own challenges in terms of **r**eal-world adaptability, dataset quality, and computational efficiency.

.

Chapter 3

SYSTEM ANALYSIS

3. SYSTEM ANALYSIS

3.1 System Analysis

System analysis is the process of examining a system to understand its components, functionality, and interactions to improve efficiency and effectiveness. It involves evaluating the system's objectives, identifying key processes, and defining the requirements necessary for optimal performance. In the context of crop disease prediction, system analysis helps in designing a model that accurately identifies plant diseases, ensuring an efficient and user-friendly experience for farmers and agricultural experts.

The proposed system utilizes deep learning techniques to enhance the accuracy of disease classification. Traditional approaches to crop disease identification relied on manual observation, which was often inaccurate and time-consuming. The introduction of computer vision and machine learning models has significantly improved the precision of disease detection. By implementing a Convolutional Neural Network (CNN) model, specifically ResNet-50, the system can process and classify plant images with high accuracy.

This project follows a structured approach, involving multiple stages such as data preprocessing, model training, prediction, and result interpretation. The system captures images of infected crops, processes them using deep learning algorithms, and provides a diagnosis. Additionally, the integration of a chatbot powered by the Gemini API enhances user interaction by providing insights into disease symptoms, prevention methods, and possible treatments.

The system architecture consists of both frontend and backend components. The frontend, developed using React.js, ensures a smooth user experience, allowing users to upload images and receive real-time results. The backend, built with Flask, handles data processing and communication with the trained model. The model itself is trained using PyTorch and utilizes a pre-trained ResNet-50 network for feature extraction.

The advancement of deep learning and cloud-based deployment allows the system to operate efficiently without requiring high-end hardware. With the ability to process large datasets and continuously improve through training, the system ensures scalability and adaptability to new crop diseases. The analysis of the system enables continuous refinement, ensuring that it remains an effective tool for the agricultural sector.

3.2 Existing System

Existing crop disease prediction systems have made significant progress with the introduction of machine learning and deep learning models, but they still face several challenges that limit their efficiency. Traditional approaches relied on manual inspection by farmers or agricultural experts, which often resulted in delayed detection and human errors. Rule-based image processing techniques were introduced to improve disease identification using color, texture, and shape analysis, but these methods lacked robustness against variations in lighting, leaf positioning, and background interference.

To address these challenges, machine learning models such as Support Vector Machines (SVM) and K-Nearest Neighbors (KNN) were employed. These models improved disease classification by learning from predefined feature sets. However, they required manual feature extraction, making them inefficient for large-scale and diverse crop datasets. The introduction of deep learning, particularly Convolutional Neural Networks (CNNs), revolutionized crop disease detection by automating feature extraction and improving accuracy. Pre-trained models like AlexNet, VGG16, and ResNet-50 have been used to classify diseases based on leaf images, showing promising results.

Despite these advancements, existing deep learning models still have certain limitations. Many models are computationally expensive and require extensive datasets for training. Some models struggle with real-time processing, making them impractical for on-field applications. Additionally, most models do not effectively integrate with user-friendly interfaces, making them less accessible to farmers

3.2.1 Disadvantages of Existing System

Vanishing Gradient Problem

Deep networks like VGG16 and AlexNet suffer from the vanishing gradient problem, where gradients become too small during backpropagation. This slows down learning and makes it difficult for earlier layers to update their weights effectively, reducing overall model performance.

High Computational Cost and Memory Consumption

Models like VGG16 and AlexNet have millions of parameters, requiring powerful hardware for training and inference. Their high memory and processing demands make them unsuitable for real-time applications and mobile deployment.

Lack of Mobile Optimization

Traditional deep learning models are not optimized for mobile or edge devices. While MobileNet improves efficiency, it sacrifices accuracy, making it difficult to balance both performance and computational constraints in resource-limited environments.

Overfitting on Small Datasets

Due to their large number of parameters, older models tend to overfit when trained on small datasets. They memorize training data instead of generalizing well, leading to poor performance on unseen images.

Slow Training and Inference Times

Deep networks require long training times and slow inference, making them impractical for real-time applications. Models like InceptionNet, despite being optimized for feature extraction, still take considerable time to train and deploy efficiently.

3.3 Proposed System

The proposed system is designed for detecting crop diseases using deep learning techniques, specifically leveraging the ResNet-50 architecture. The system aims to help farmers accurately diagnose crop diseases by analyzing leaf images, facilitating timely intervention and effective disease management. Unlike traditional manual disease identification methods, which are time-consuming and require expert knowledge, this system automates the process using advanced computer vision techniques and artificial intelligence.

This system follows a structured pipeline that includes image preprocessing, disease classification, and chatbot-based assistance, ensuring an efficient and user-friendly approach to plant disease diagnosis. The model is trained on the New Plant Diseases Dataset, enabling it to classify **38 different plant disease categories**, covering various crops and their respective diseases. These include:

- **Tomato diseases:** Early Blight, Late Blight, Leaf Mold, Septoria Leaf Spot, Target Spot, Tomato Yellow Leaf Curl Virus, Tomato Mosaic Virus, Bacterial Spot, and Healthy Tomato Leaves.
- **Apple diseases:** Apple Scab, Black Rot, Cedar Apple Rust, and Healthy Apple Leaves.
- **Grape diseases:** Black Rot, Esca (Black Measles), Leaf Blight (Isariopsis Leaf Spot), and Healthy Grape Leaves.
- **Potato diseases:** Early Blight, Late Blight, and Healthy Potato Leaves.
- **Corn (Maize) diseases:** Northern Leaf Blight, Common Rust, Gray Leaf Spot, and Healthy Corn Leaves.
- **Peach disease:** Bacterial Spot and Healthy Peach Leaves.
- **Strawberry disease:** Leaf Scorch and Healthy Strawberry Leaves.

- **Bell Pepper disease:** Bacterial Spot and Healthy Bell Pepper Leaves.
- **Soybean condition:** Healthy Soybean Leaves.

By integrating deep learning-based classification with a chatbot-powered guidance system, the proposed solution provides an end-to-end framework that supports farmers with **real-time insights and actionable recommendations**, ensuring effective crop health management.

The system can be broken down into five main components:

- Image Acquisition and preprocessing
- Disease Classification Using ResNet-50
- Chatbot-Based Farmer Assistance
- User Interface for Image Upload and Result Display
- Backend processing and System Integration

Image Acquisition and Preprocessing:

The system begins by acquiring leaf images from users through an intuitive interface. These images undergo preprocessing techniques such as resizing, normalization, and noise reduction. Background elimination techniques are applied to ensure that only the leaf portion is analyzed, reducing unwanted variations in the dataset and improving classification accuracy.

Disease Classification Using ResNet-50:

The core of the system is the ResNet-50 model, which classifies diseases based on deep learning techniques. With its 50-layer residual network, ResNet-50 efficiently extracts features from images, distinguishing healthy leaves from diseased ones. This ensures accurate predictions while optimizing computational efficiency, making it suitable for real-time analysis.

Chatbot-Based Farmer Assistance:

The chatbot is also embedded in the interface for additional guidance and suggestions, ensuring a seamless experience for users.

User Interface for Image Upload and Result Display:

A user-friendly interface allows farmers to upload leaf images easily. Once analyzed, the results, including disease name and treatment recommendations, are displayed clearly. The chatbot is also embedded in the interface for additional guidance, ensuring a seamless experience for users.

Backend Processing and System Integration:

The backend system handles the communication between the front-end interface, the deep learning model, and the chatbot. It efficiently processes user inputs, runs the classification model, retrieves relevant disease information, and delivers responses in real time. This ensures smooth interaction between all system components, making the application scalable and reliable.

3.3.1 Advantages of Proposed System

High Accuracy in Disease Detection

By using ResNet-50, the system ensures precise classification of plant diseases. Its deep residual learning helps in extracting fine-grained features, making it more reliable compared to traditional deep learning models.

Efficient Feature Extraction

ResNet-50's residual connections allow deeper network training without performance degradation. This enhances feature extraction, enabling the model to identify diseases even from images with variations in lighting, angle, or background.

Efficiency

The success rate of 96.83% shows the efficiency of the presented methodology. The proposed system accesses the desktop virtually with maximum efficiency.

Automated Assistance Through Chatbot

The integrated chatbot provides instant guidance on disease remedies, reducing farmers' dependency on external resources. This ensures quick decision-making and easy access to treatment solutions without manual research.

No Need for Disease Data Storage

Since the system does not store disease-related data, it eliminates concerns related to data management and privacy. The model works on live input images without requiring a database of previously detected diseases.

Scalable and Upgradable

Although the system does not store disease data, it can be improved over time by retraining the ResNet-50 model with new datasets. This allows the system to adapt to emerging plant diseases without major structural changes.

3.4 Functional Requirements

Functional Requirements talks about a functionality that need to be carried out by a system or its component. A functionary is considered as a specification of behavior between outputs and inputs.The Users or Actors identified in Project are Admin and user. The different functionalities of the corresponding Users are tabulated as shown in **Table**

ACTOR	FUNCTIONALITIES
Farmer (User)	Upload plant leaf images for disease detection. View disease classification results and confidence score. Access remedy suggestions for identified diseases. Interact with the chatbot for additional guidance.
System	Process uploaded images through preprocessing techniques. Perform disease classification based on trained deep learning model. Provide the predicted disease label with a confidence score.
Chatbot	Respond to user queries regarding detected diseases Provide disease descriptions, symptoms, and causes. Manage user queries and feedback for system improvements.

Table 3.4.1: User and Functionalities

3.5 Non-Functional Requirements

Non-functional Requirements states criteria that can be used to evaluate the operation of a system, rather than specific behaviors. They are different from Functional Requirements which define specific behavior or functionalities.

These are the quality constraints that our system satisfies according to the project

Performance Efficiency

The system should process and classify images within a few seconds to ensure quick responses. The chatbot should also provide near-instant replies to user queries without noticeable delays.

Scalability

The architecture should support future improvements, such as expanding the dataset, retraining the model, or enhancing chatbot capabilities without requiring significant modifications.

Usability

The interface should be simple and intuitive, allowing farmers and non-technical users to interact with the system easily. Clear instructions and minimal steps should be required for image uploads and chatbot interactions.

Maintainability

The system should be modular, making it easier to update or improve individual components, such as the chatbot, image processing pipeline, or ResNet-50 model, without affecting overall functionality.

Accuracy and Robustness

The model should be trained to handle variations in lighting, background, and image quality while maintaining high accuracy in disease classification. The chatbot should provide relevant and precise remedy suggestions.

3.6 Hardware Requirements

Any software application defines the most common requirements known as the physical computer resources or hardware, A hardware requirements list regularly come with hardware compatibility list (HCL), particularly in case of operating systems. An HCL contains tested, compatible, and sometimes incompatible hardware devices for a particular operating system or application.

The system is designed to run efficiently on Google Colab, eliminating the need for a high-end processor. A minimum of 4GB RAM is required to ensure smooth execution of tasks, while 128GB of ROM provides sufficient storage for data processing and model execution. The system leverages the Tesla T4 GPU available in Google Colab to accelerate computations, making it suitable for AI and deep learning applications. This setup ensures optimal performance without requiring extensive local hardware resources.

Processor	Google Colab is used, does not require a high-end processor.
RAM	4 GB ram or higher.
ROM	128 GB
GPU	Tesla T4

Table 3.6.1.1: Hardware Requirements

3.7 Software Requirements

Software requirements involve defining software resource requirements that need to be installed on a computer to provide optimal functioning of an application. These requirements are usually not included in the software installation package and required to be installed separately before the software is installed

Operating System	**:**	Window XP/Windows10, Linux Windows Vista/Windows 7
Languages	**:**	Python, Java Script
Python Libraries	**:**	TensorFlow/Keras, Flask, react.js, Fetch API
IDE used	**:**	Google Colab, Visual Studio
API	**:**	Gemini API

3.7.1 PYTHON

Python is an easy to learn, powerful programming language. It has efficient high- level data structures and a simple but effective approach to object-oriented programming. Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms. The Python interpreter and the extensive standard library are freely available in source or binary form for all major platforms from the Python Web site, https://www.python.org/, and may be freely distributed. The same site also contains distributions of and pointers to many free third-party Python modules, programs and tools, and additional documentation.

The Python interpreter is easily extended with new functions and data types implemented in C or C++ (or other languages callable from C). Python is also suitable as an extension language for customizable applications. This tutorial introduces the reader informally to the basic concepts and features of the Python language and system. It helps to have a Python interpreter handy for hands-on experience, but all examples are self-contained, so the tutorial can be read off-line as well. For a description of standard objects and modules, see library-index. reference-index gives a more formal definition of the language. To write extensions in C or C++, read extending- index and c- api-index.

There are also several books covering Python in depth. If you do much work on computers, eventually you find that there's some tasks you'd like to automate. For example, you may wish to perform a search-and-replace over a large number of text files, or rename and rearrange a bunch of photo files in a complicated way. Perhaps you'd like to write a small custom database,

or a specialized GUI application, or a simple game. If you're a professional software developer, you may have to work with several C/C++/Java libraries but find the usual write/compile/test/re-compile cycle is too slow. Perhaps you're writing a test suite for such a library and find writing the testing code a tedious task. Or maybe you've written a program that could use an extension language, and you don't want to design and implement a whole new language for your application. Python is just the language for you. You could write a Unix shell script or Windows batch files for some of these tasks, but shell scripts are best at moving around filesand changing text data, not well- suited for GUI applications or games. You could write a C/C++/Java program, but it can take a lot of development time to get even a first-draft program. Python is simpler to use, available on Windows, Mac OS X, and Unix operating systems, and will help you get the job done more quickly.

Python comes with a large collection of standard modules that you can use as the basis of your programs or as examples to start learning to program in Python. Some of these modules provide things like file I/O, system calls, sockets, and even interfaces to graphical user interface toolkits like Tk Python is an interpreted language, which can save you considerable time during program development because no compilation and linking is necessary. The interpreter can be used interactively, which makes it easy to experiment with features of the language, to write throw-away programs, or to test functions during bottom-up program development. Python enables programs to be written compactly and readably. Programs written in Python are typically much shorter than equivalent C, C++, or Java programs, for several reasons.

Python is extensible: if you know how to program in C it is easy to add a new built-in function or module to the interpreter, either to perform critical operations at maximum speed, or to link Python programs to libraries that may only be available in binary form (such as a vendor- specific graphics library).

3.7.1.1 APPLICATIONS OF PYTHON

Web Development

It is one of the most astonishing applications of Python. This is because Python comes up with a wide range of frameworks like Django, Flask, Bottle, and a lot more that provide ease to developers. Furthermore, Python has inbuilt libraries and tools which make the web development process completely effortless. Use of Python for web development also offers:

- Amazing visualization
- Convenience in development
- Enhanced security

- Fast development process

Machine Learning and Artificial Intelligence

Machine Learning and Artificial Intelligence are the hottest subjects right now. Python along with its inbuilt libraries and tools facilitate the development of AI and ML algorithms. Further, it offers simple, concise, and readable code which makes it easier for developers to write complex algorithms and provide a versatile flow. Some of the inbuilt libraries and tools that enhance AI and ML processes are:

- NumPy for complex data analysis
- Keras for Machine learning
- SciPy for technical computing
- Seaborn for data visualization

Data Science

Data science involves data collection, data sorting, data analysis, and data visualization. Python provides amazing functionality to tackle statistics and complex mathematical calculations. The presence of in-built libraries provides convenience to data science professionals. Some of the popular libraries that provide ease in the data science process are TensorFlow, Pandas, and Socket learning. These libraries provide an ecosystem for fine-tuning data models, data pre-processing, and performing complex data analysis.

3.7.2 AI-BASED PLANT DISEASE DIAGNOSIS AND PRECISION AGRICULTURE

Artificial Intelligence (AI) is transforming agriculture by enabling automated disease diagnosis and precision farming solutions. The integration of deep learning models, such as ResNet-50, in plant disease detection helps farmers identify crop diseases at an early stage, minimizing yield loss and improving food security.

AI-driven plant disease detection leverages computer vision techniques to analyze leaf images, classify diseases, and suggest treatments. The system processes images through multiple layers of neural networks to extract meaningful features and detect patterns that indicate disease symptoms.

Common AI-based agricultural solutions include:

- Deep Learning Based Disease Identification
- IoT-Enabled Smart Farming Systems
- Drones and Satellite Imagery for Crop Monitoring

- Automated Irrigation and Fertilization Systems
- AI Chatbots for Agricultural Assistance

The evolution of AI in agriculture has been marked by the integration of machine learning and image processing technologies. In the early 2000s, research focused on developing machine learning models capable of identifying plant diseases based on predefined feature extraction techniques. However, these methods had limitations in handling complex patterns and variations in leaf images. With the advent of deep learning, particularly Convolutional Neural Networks (CNNs), plant disease detection has reached new levels of accuracy and efficiency. ResNet-50, a widely used deep learning model, plays a crucial role in this transformation by using residual learning to extract intricate patterns from images, ensuring more precise disease classification.

To ensure the reliability of AI-powered plant disease detection, the system incorporates various components working in synchronization. The first step in the process is image preprocessing, where raw leaf images undergo resizing, normalization, and noise reduction to enhance their quality. This helps in minimizing distortions and variations, enabling the model to make more accurate predictions. Once the images are processed, they are fed into the ResNet-50 model, which analyzes them layer by layer to identify the presence of disease symptoms. The classification results are then displayed to the user through an interactive interface, making it easy for farmers to interpret the findings.

Beyond disease classification, AI-powered agricultural systems also provide farmers with essential guidance on disease management. A chatbot, integrated into the system, serves as a virtual assistant that helps farmers understand the detected diseases and suggests appropriate remedies. The chatbot utilizes natural language processing (NLP) to interpret user queries and provide relevant responses, eliminating the need for farmers to conduct extensive external research. Additionally, the backend infrastructure of the system ensures smooth communication between the user interface, the deep learning model, and the chatbot, making the entire process seamless and efficient.

AI-based plant disease detection represents a significant advancement in precision agriculture. By combining deep learning, image processing, and interactive chatbot assistance, this technology empowers farmers with accurate and timely information, helping them make informed decisions about disease management. As AI continues to evolve, its role in agriculture is expected to expand, leading to more sophisticated and efficient farming solutions that enhance productivity while reducing crop losses.

3.8 Why Python:

Python is chosen as the primary programming language for this project due to its simplicity, flexibility, and extensive support for artificial intelligence (AI) and computer vision applications. Python's syntax is clean and easy to understand, which speeds up development and reduces the chances of errors. This makes it a preferred choice for researchers and developers working on deep learning and image processing tasks.

One of the biggest advantages of Python is its rich ecosystem of libraries. For deep learning, it offers frameworks like TensorFlow, PyTorch, and Keras, which provide pre-built functions for neural networks, training models, and optimizing performance. These frameworks simplify the implementation of the ResNet-50 model, which is used in this project for plant disease classification.

Python's versatility also allows seamless integration with databases, cloud platforms, and web-based interfaces. This ensures that the system can scale efficiently, enabling future improvements such as detecting a wider range of plant diseases or incorporating real-time monitoring. The language's support for natural language processing (NLP) through libraries like NLTK and spaCy makes it easier to integrate the chatbot, which provides farmers with actionable insights on disease treatment and prevention.Another reason Python is ideal for this project is its strong support for data handling and analysis. With libraries like NumPy and Pandas, large datasets of plant leaf images can be processed efficiently, allowing smooth model training and accurate predictions. Additionally, Python's Matplotlib and Seaborn help in data visualization, making it easier to analyze trends and performance metrics.

Furthermore, Python has a vast community of developers and researchers who continuously contribute to its growth. This means that any technical challenges encountered during development can be quickly resolved with the help of extensive documentation, tutorials, and online forums. Overall, Python's ease of use, robust library support, scalability, and strong AI capabilities make it the perfect choice for implementing this plant disease detection system.

Why Pytorch:

PyTorch is an open-source deep-learning framework developed by Meta AI, widely used in research and production for its flexibility, ease of use, and powerful features. It provides dynamic computation graphs, allowing developers to modify and debug models effortlessly compared to static graph-based frameworks. This makes PyTorch an excellent choice for deep learning applications, including computer vision, which is the primary focus of this project. One

of the key advantages of PyTorch is its seamless integration with GPUs, enabling accelerated model training and inference. By simply using .to(device), computations can be shifted between CPU and GPU without modifying the core code, making it highly efficient for handling large-scale datasets.

Another reason for choosing PyTorch is its strong support for computer vision tasks through the Torchvision library, which provides pre-trained models, datasets, and image transformation utilities. In this project, ResNet-50, a deep convolutional neural network from Torch vision, is utilized for image classification. PyTorch's Autograd engine further simplifies the model training process by automatically computing gradients during backpropagation, eliminating the need for manual differentiation. This ensures that optimization algorithms like Stochastic Gradient Descent (SGD) or Adam function efficiently without additional complexity. The ecosystem of PyTorch is also vast, with strong community support and integration with libraries like NumPy, OpenCV, and Flask. These features make it a suitable choice for real-world deep-learning applications, including deploying AI models in web applications.

In this project, PyTorch plays a crucial role in training and fine-tuning the ResNet-50 model for classifying plant diseases. It facilitates image processing, model optimization, and evaluation using metrics such as confusion matrices and classification reports. Additionally, the trained model is integrated into a Flask API, allowing real-time image predictions. The ability of PyTorch to handle both research-oriented experiments and production-level deployments makes it an essential tool for building advanced machine-learning models with high efficiency and performance.

Why Flask for Backend?

Flask is a lightweight and flexible web framework for Python, widely used for building web applications and APIs. It is a micro-framework, meaning it provides essential tools for web development without unnecessary complexity. Unlike larger frameworks like Django, Flask offers greater control over application structure and customization, making it ideal for projects requiring specific integrations and optimizations.

For this project, Flask is used to create an API that allows seamless interaction between the trained PyTorch model and the frontend application. The API receives image inputs, processes them using ResNet-50, and returns classification results in real time. Flask’s request handling capabilities enable smooth data transmission between the client and server, ensuring efficient communication. Additionally, it supports JSON responses, making it easy to send structured

output that can be processed by web or mobile applications.

One of the key reasons for choosing Flask is its compatibility with machine learning and deep learning models. Since PyTorch models can be loaded and executed in Python environments, Flask provides an efficient way to serve these models as RESTful APIs. Moreover, Flask integrates well with Flask-CORS, allowing cross-origin requests, which is essential when the frontend and backend run on different servers.

Security and authentication are also handled efficiently in Flask using Flask-JWT-Extended, which enables token-based authentication for secure access to the API. Additionally, Flask works well with PyMongo, making it easy to store and retrieve classified results in a MongoDB database. These features collectively make Flask an excellent choice for deploying AI-based applications with backend processing.

Why NumPy:

In Python we have lists that serve the purpose of arrays, but they are slow to process. NumPy aims to provide an array object that is up to 50x faster than traditional Python lists. The array object in NumPy is called ND array, it provides a lot of supporting functions that make working with ND array very easy. Arrays are very frequently used in data science, where speed and resources are very important.

Why ResNet-50:

ResNet-50 was chosen for this crop disease detection project because of its superior ability to handle deep learning tasks efficiently while overcoming the vanishing gradient problem that typically affects very deep neural networks. Unlike traditional convolutional neural networks (CNNs) such as VGG and AlexNet, which suffer from performance degradation as layers increase, ResNet-50 incorporates residual learning through shortcut connections. These residual connections allow the network to retain essential features from earlier layers while ensuring stable gradient propagation during backpropagation. This architecture significantly enhances learning capabilities, making it ideal for complex image classification tasks like identifying crop diseases.

Another key reason for selecting ResNet-50 is its balance between depth and computational efficiency. With 50 layers, it is deep enough to capture intricate patterns in plant leaf images, yet it remains computationally feasible compared to larger networks such as ResNet-

101 or ResNet-152. VGG and AlexNet, although successful in earlier image recognition tasks, are comparatively shallow and require more parameters, leading to higher computational costs and increased risk of overfitting when dealing with large datasets. ResNet-50, on the other hand, achieves better accuracy with fewer parameters due to its innovative residual blocks, making it a more suitable choice for real-world applications where efficiency and accuracy are both crucial.

Additionally, ResNet-50 is widely adopted in transfer learning, making it an excellent choice for crop disease detection, especially when working with limited labelled data. Since ResNet-50 has been pre-trained on large-scale image datasets like ImageNet, it can leverage learned features and adapt them to agricultural images with minimal fine-tuning. This not only reduces training time but also improves model generalization, ensuring that the network can accurately identify a variety of crop diseases even with a relatively smaller dataset. In contrast, older architectures like VGG and AlexNet require more extensive training and manual fine-tuning to achieve comparable performance.

Overall, the decision to use ResNet-50 in this project stems from its deep yet efficient architecture, superior feature extraction capabilities, and robust performance in transfer learning scenarios. These advantages make it well-suited for addressing the complexities of crop disease detection, ensuring high accuracy while maintaining computational efficiency.

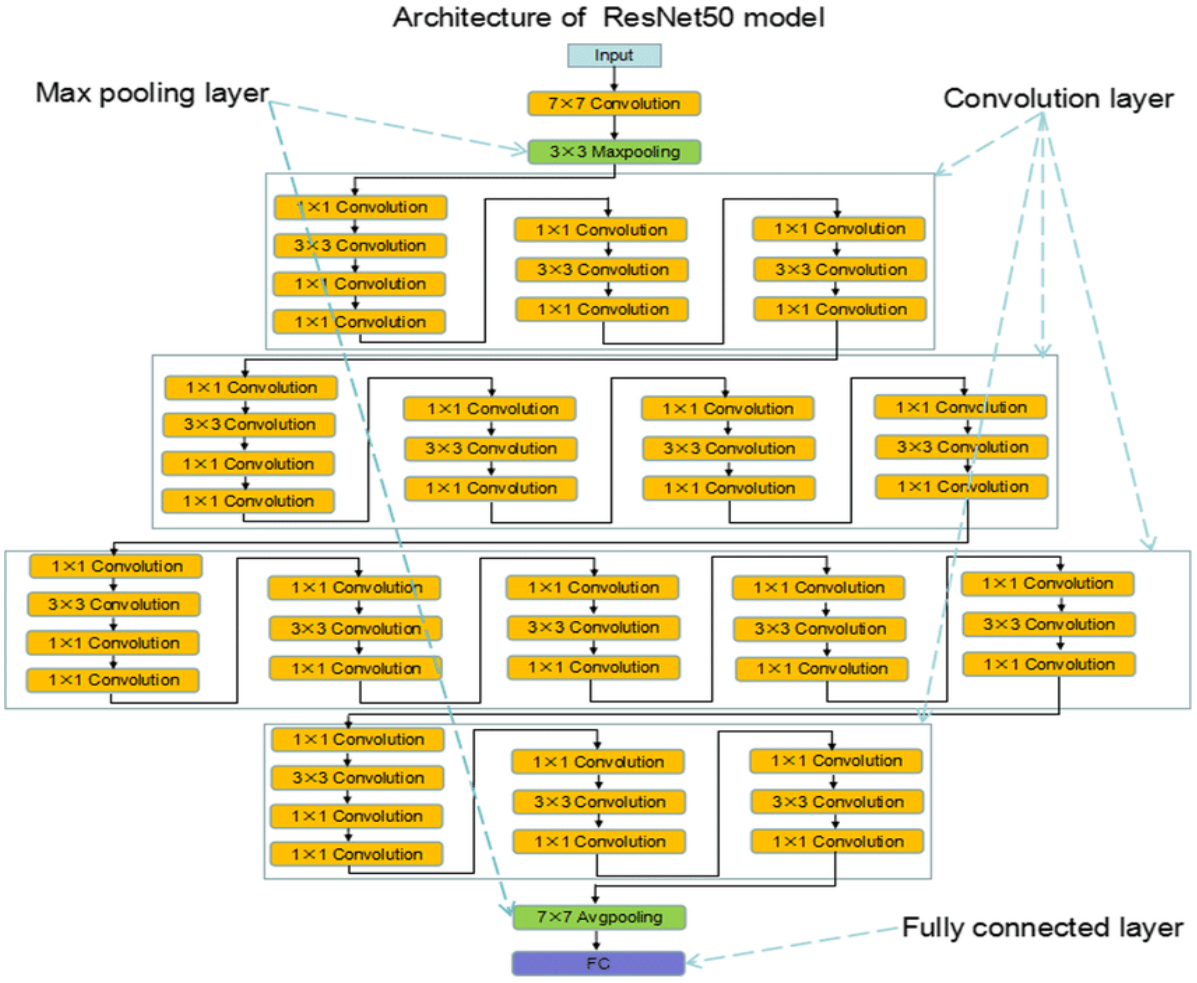


Figure 3.8.1 ResNet Architecture

3.9 Pre-built AI Solutions in Google Generative AI and PyTorch:

Solutions Google Generative AI and PyTorch offer various pre-trained models and solutions that can be leveraged for different applications in deep learning and AI-powered projects. These models are optimized for tasks like image classification, object detection, text generation, and embeddings. Below are some of the key solutions:

- **Image Classification (Vision Model)** – Recognizes and categorizes objects in images.
- **Object Detection** – Identifies multiple objects in an image along with their locations.
- **Text Generation (LLMs like Gemini, GPT, etc.)** – Generates human-like text based on the given prompt.
- **Embeddings for Similarity Matching** – Converts words, sentences, or images into vector representations for comparisons.
- **Question-Answering** – Extracts answers from a given document or context.

- **Summarization** – Generates concise summaries of long texts.

These solutions are pre-built and can be fine-tuned for specific use cases. In your project, models from Google Generative AI and PyTorch provide powerful classification capabilities and ensure efficient deep-learning execution

3.10 CNN (Convolutional Neural Networks):

Convolutional Neural Networks (CNNs or ConvNETs) are Deep Learning algorithms that process images, assign importance to objects in the image using learnable weights and biases, and can differentiate images from each other. A convolutional neural network is essentially a neural network that uses a convolution layer and pooling layer. The convolutional layer convolves into a smaller areato extract features, while the pooling layer picks the data with the highest value within an area. They require less pre-processing in comparison to other classification algorithms and are able to learn filters and characteristics. The architecture of convolutional neural networks was based on the organization of the visual cortex.

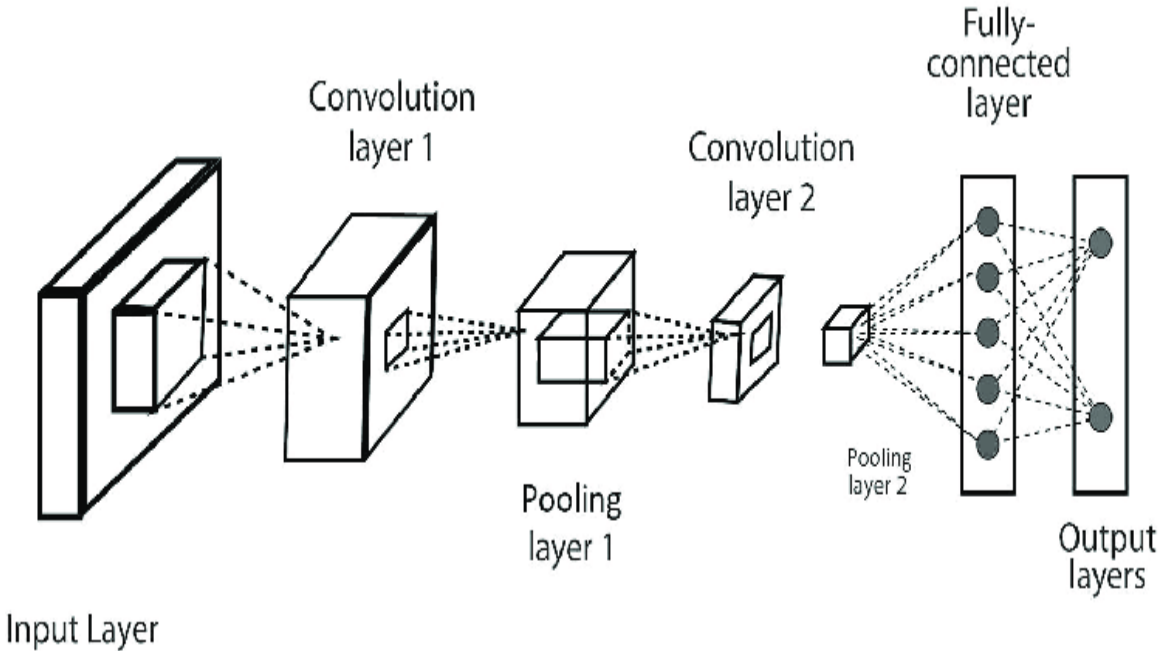


Figure 3.10.1: Convolutional Neural Networks

Detailed Processing of CNN in the System

3.10.1 Data Pre-processing Image Sampling

The dataset used for training the crop disease prediction system consists of images of various diseased and healthy crop leaves. The original image dimensions varied significantly, requiring a standardized preprocessing pipeline. Each image was resized to 224x224 pixels, ensuring compatibility with the ResNet model. This resizing step helped maintain uniformity

across the dataset, enabling efficient training and inference. To enhance the model's ability to generalize across different lighting conditions and angles, data augmentation techniques such as rotation, flipping, contrast adjustments, and Gaussian noise addition were applied. These augmentations enriched the dataset, reducing overfitting and improving model robustness.

Feature Extraction using ResNet - 50

The ResNet architecture was chosen for its superior feature extraction capabilities, overcoming vanishing gradient issues present in deep networks. The implemented model leveraged ResNet-50, which consists of 50 layers, including convolutional layers, batch normalization layers, and skip connections. The key advantage of ResNet lies in its residual learning framework, where identity mappings allow gradient flow through deep layers without degradation. This enabled effective training of the network on the crop disease dataset.

The input image passed through an initial convolutional layer, followed by multiple residual blocks that captured hierarchical features of leaf textures, color variations, and disease patterns. The final feature map was passed through a global average pooling layer, reducing dimensionality while preserving the most significant features. A fully connected layer with a softmax activation function classified the image into different disease categories, providing probabilities for each class.

Model Training and Optimization

The training process involved feeding the preprocessed dataset into the ResNet model using a batch size of 32. The Adam optimizer was employed with an initial learning rate of 0.001, which was dynamically reduced based on validation loss improvement. Cross-entropy loss was used as the objective function to optimize classification performance. To prevent overfitting, techniques such as dropout layers and L2 regularization were incorporated. The model was trained for 50 epochs, with real-time validation to monitor performance. After training, the best model was selected based on accuracy and loss metrics.

3.11 SUMMARY

Crop Disease Detection using a Convolutional Neural Network (CNN) enhances agricultural efficiency by enabling early detection of plant infections. This system processes captured leaf images through image sampling, dimensionality reduction, and clustering techniques to improve classification accuracy. The CNN model identifies diseases based on visual symptoms and maps them to predefined categories, aiding farmers in timely decision-making. By leveraging deep learning, this technology minimizes crop losses, optimizes yield, and supports sustainable farming. As advancements continue, AI-driven disease prediction will revolutionize precision agriculture, providing farmers with real-time, automated diagnostics for better crop management.

Chapter 4
SYSTEM DESIGN

4. SYSTEM DESIGN

4.1 Introduction

System Design is a process through which requirements are converted into a structured representation of software. Initially, this representation provides a high-level overview of the system. With subsequent improvements, the design evolves into a refined structure that closely resembles the final implementation. A well-defined system design ensures that the software can be efficiently tested, maintained, and optimized. It serves as the foundation for all subsequent software engineering and maintenance phases.

The input design of the system follows a simple yet effective approach, allowing users to interact with the model seamlessly. The user only needs access to a smartphone or computer with an internet connection to upload crop leaf images and query the chatbot. The system is designed to be intuitive, providing farmers and agricultural experts with an effortless way to diagnose plant diseases and receive relevant information.

4.2 System Architecture

Architecture of the System is the conceptual model explaining the arrangement of the various components that makes the System. The Overall Architecture of the System which is shown in fig 4.2 encompasses the main modules Image pre-processing, features extraction from the pre- processing, System training for training the system with multiple images of hands and Databases corresponding different information and nature of hands, System testing for testing the progress of the system to identify hands which is given as input.

The system architecture is designed to facilitate plant disease detection and provide remedies using a chatbot. It consists of a frontend built with React, a Flask-based backend, a deep learning model (ResNet-50) for image classification, and the Gemini API for NLP-based chatbot responses. Each component plays a crucial role in ensuring seamless interaction between the user and the system.

The user interacts with the system through the React-based frontend, where they can either upload an image of a plant to check for diseases or ask a query regarding plant health. When an image is uploaded, it is sent to the Flask backend, which processes the request and forwards it to the deep learning model for analysis. The ResNet-50 model extracts important features from the image and classifies the plant disease based on the learned patterns. The classification result is then sent back to the backend and displayed on the frontend. For text-based queries, the backend forwards the user's question to the Gemini API, which utilizes Natural Language Processing (NLP) to generate an appropriate response. The chatbot processes the query, fetches relevant information, and sends the response back to the backend. The backend then returns the AI-

generated response to the frontend, where the user can view the suggested remedies or general agricultural advice.

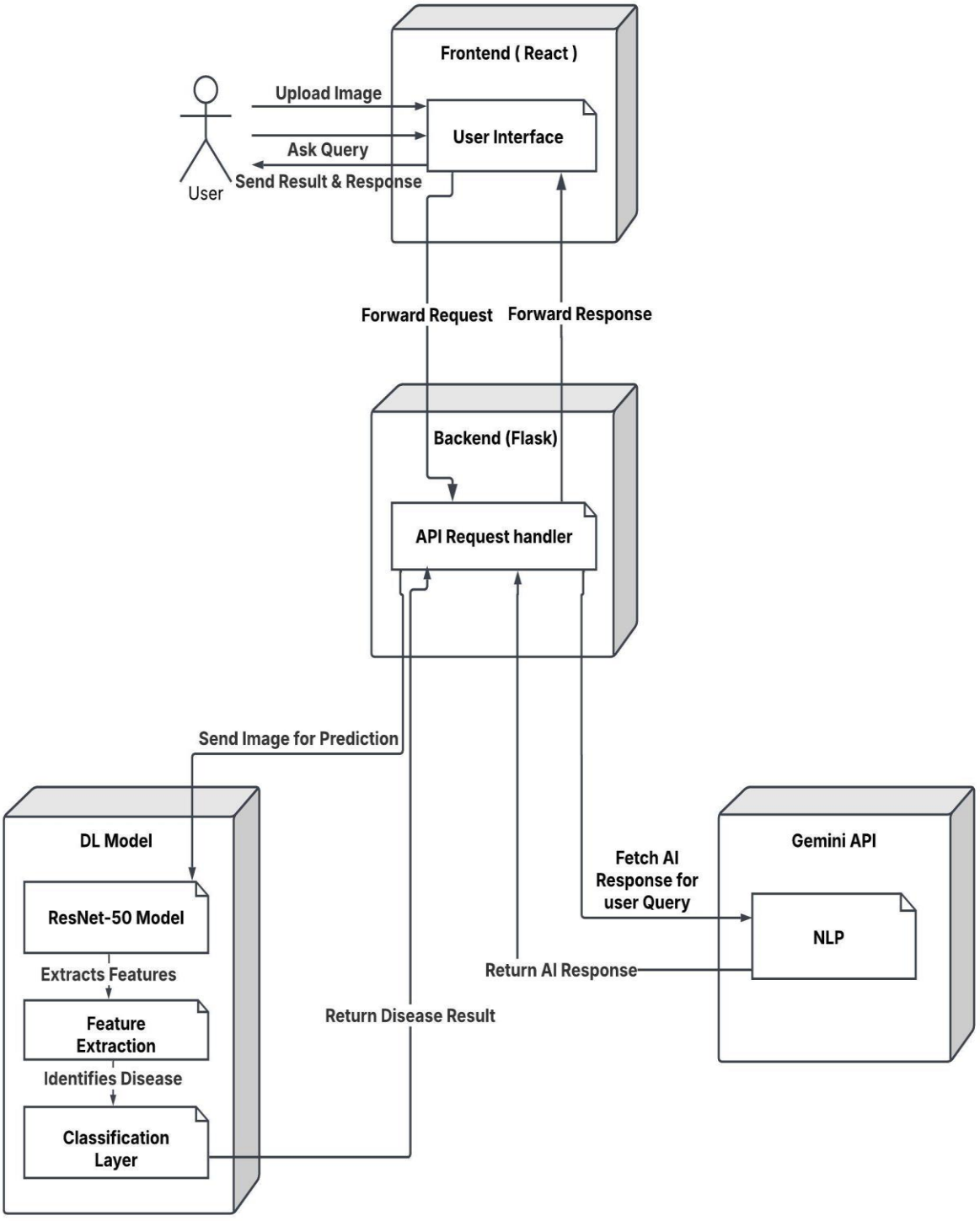


Figure 4.2.1: System Architecture

4.2.1 CNN System Architecture

The ConvNet's job is to compress the images into a format that is easier to process while preserving elements that are important for obtaining a decent prediction. This is critical for designing an architecture that is capable of learning features while also being scalable to large datasets. A convolutional neural network, ConvNets in short has three layers which are its building blocks, let's have a look:

4.2.1.1 **Convolutional Layer** (CONV)

They are the foundation of CNN, and they are in charge of executing convolution operations. The Kernel/Filter is the component in this layer that performs the convolution operation (matrix). Until the complete image is scanned, the kernel makes horizontal and vertical adjustments dependent on the stride rate. The kernel is less in size than a picture, but it has more depth. This means that if the image has three (RGB) channels, the kernel height and width will be modest spatially, but the depth will span all three

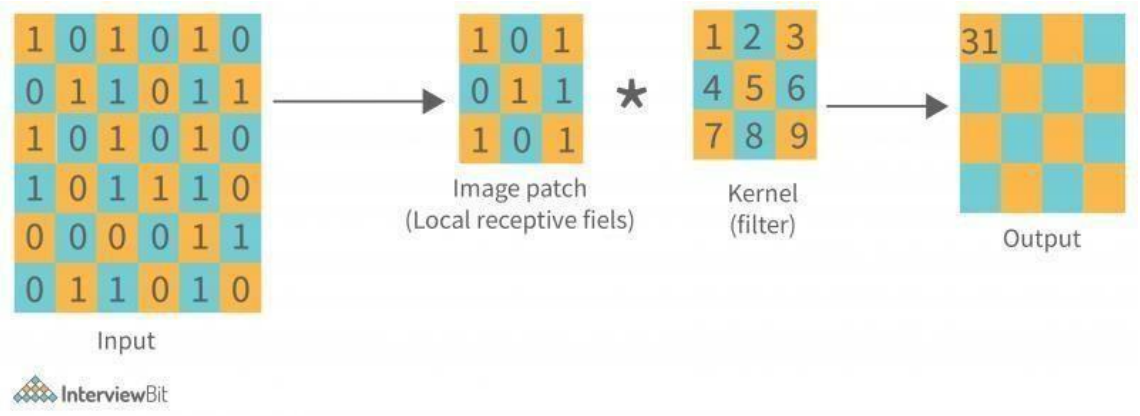


Figure 4.2.1.1: Convolutional Layer

4.2.1.2 **Pooling Layer** (POOL)

This layer is in charge of reducing dimensionality. It aids in reducing the amount of computing power required to process the data. Pooling can be divided into two types: maximum pooling and average pooling. The maximum value from the area covered by the kernel on the image is returned by max pooling. The average of all the values in the part of the image covered by the kernel is returned by average pooling.

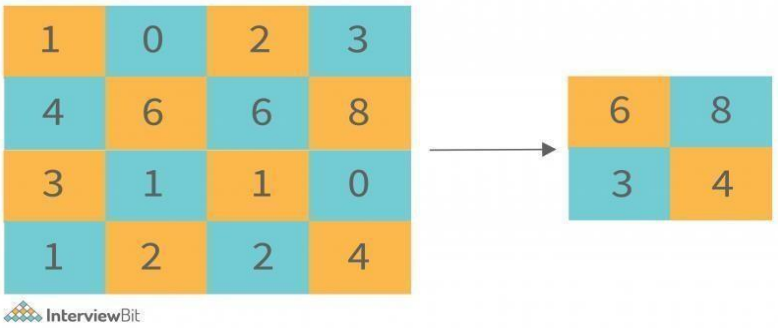


Figure 4.2.1.2: Pooling Layer

4.2.1.3 **Fully Connected Layer** (FC)

The fully connected layer (FC) works with a flattened input, which means that each input is coupled to every neuron. After that, the flattened vector is sent via a few additional FC layers, where the mathematical functional operations are normally performed. The classification procedure gets started at this point. FC layers are frequently found near the end of CNN architectures if they are present.

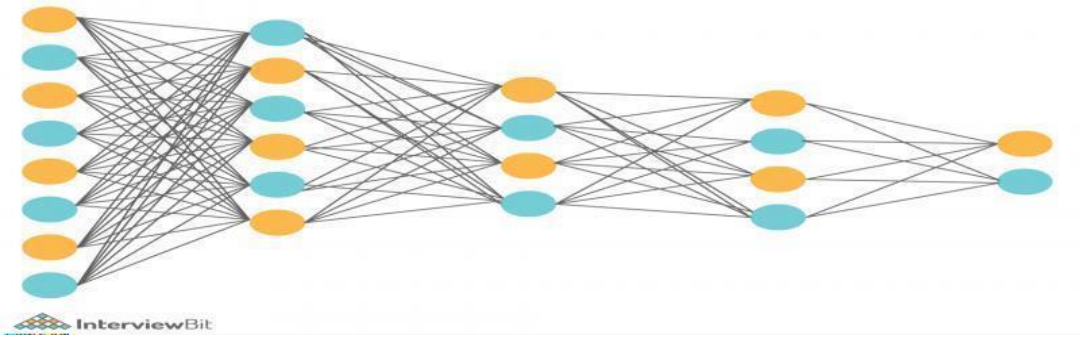


Figure 4.2.1.3: Fully Connected Layer

4.3 UML Diagrams used in Design

Design is the first step in the development phase for an engineered product or system. Design is the place where quality is fostered in software development. Design is the only way that we can accurately translate a user's requirements into a finished software product or system. Software design serves as the foundation for all software engineers and software maintenance steps that follow. Without design, we risk building an unstable design one that will fail when small changes are made, one that may be difficult to test, and one whose quantity cannot be accessed until late in the software engineering process.

Taking software requirements specification document of analysis phase as input to the design phase we have drawn Unified Modelling Language (UML) diagrams. UML depends on the visual modelling of the system. Visual modelling is the process of taking the information from the model and displaying it graphically using some sort of standards set of graphical elements.

UML Diagrams are drawn using the Star UML Diagrammed Software. Complexity is better understood when it is displayed visually rather than written textually. By producing visual models of a system, one can understand how system works on several levels and can model the interactions between the users and the system. Each UML diagram is designed to let developers and customers view a software system from a different perspective and in varying degrees of abstraction.

4.4 Use case diagram

Use Case Diagram captures the system's functionality and requirements by using actors and use cases. Use Cases model the services, tasks, function that a system needs to perform. It models how an external entity interacts with the system to make it work. Use cases represent high-level functionalities and how a user will handle the system. Actors can be defined as something that interacts with the system. Actors can be a human user, some internal applications, or may be some external applications. When we are planning to draw a Use Case diagram, we should have the following items identified.

Functionalities to be represented as use case

- Actors
- Relationships among the use cases and actors.

There can be following five Associations in Use case diagrams

- Association between actor and use case
- Generalization of an actor
- Extend between two use cases
- Include between two use cases Generalization of a use case

4.4.1 Scenarios

A Scenario is a specific sequence of actions and interactions between actors and the system; it is also called a use case instance. It is one particular story of using a system, or one path through the use case; for example, the scenario of successfully purchasing items with cash, or the scenario of failing to purchase items because of a credit payment denial. Informally then, a use case is a collection of related success and failure scenarios that describe an actor using a system to support a goal.

4.4.2 Use case diagram representing overview of the System

Actors identified are Admin and User. Associations may be unidirectional or bidirectional. An undirected association relationship is an association that is navigable in only one direction and in which the control flows from one classifier to another only one of the association ends specifies navigability. The association is represented by a line with arrow at one end. A bidirectional association relationship is an association that is navigable both directions and in which the control flows from both classifiers to another. The association is represented by a normal line between use case and an actor.

There are 2 actors present in the diagram

I. USER

- Uploads Image.
- Receive Disease Prediction.
- Chat with Chatbot

II. System

- Process Image & Predict Disease
- Provide Chatbot Responses

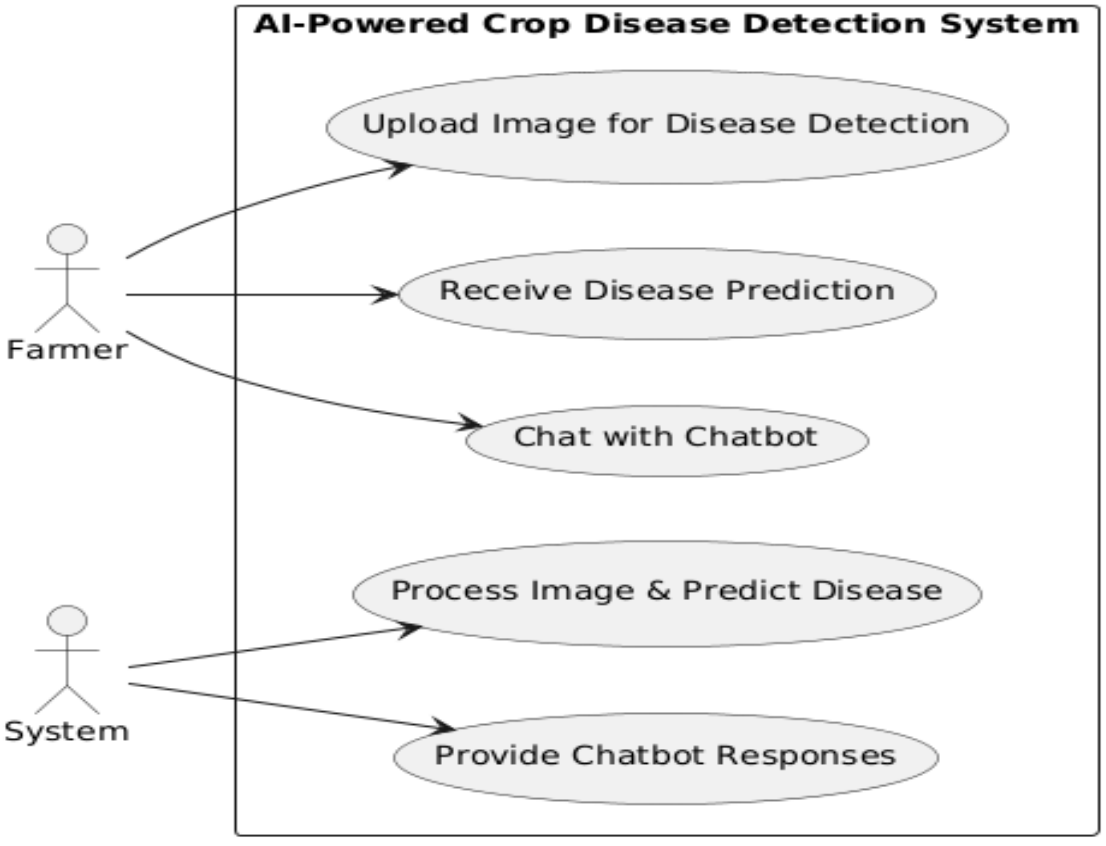


Figure 4.4.1: Overview of the System – (Use case diagram)

4.5 Class Diagram:

Class Diagrams helps to design the structural model of the system by showing classes, attributes, operations and relationships between classes. `The Class Diagram shown in fig: 5.3.2.1 displays the classes that represent different users with their respective attributes and functionalities. It also displays classes that represent different services provided by the Proposed System.

Class: Farmer

Here it contains the information related to front end that user can interact with:

-Upload Image ()

-receive Prediction ()

-Chat With Chatbot ()

Class: System

Here it contains the information related to the system functionalities

-Process Image ()

-Predict Disease ()

-Provide Chatbot Response ()

Class: Image Processor

Here it contains the information related to Image processing

-analyze Image ()

-detect Disease ()

Class: Disease Prediction

Here it contains the information related to Disease Prediction Model

-generate Prediction ()

Class: Chatbot

Here it contains the information related to the Chatbot

-get Response ()

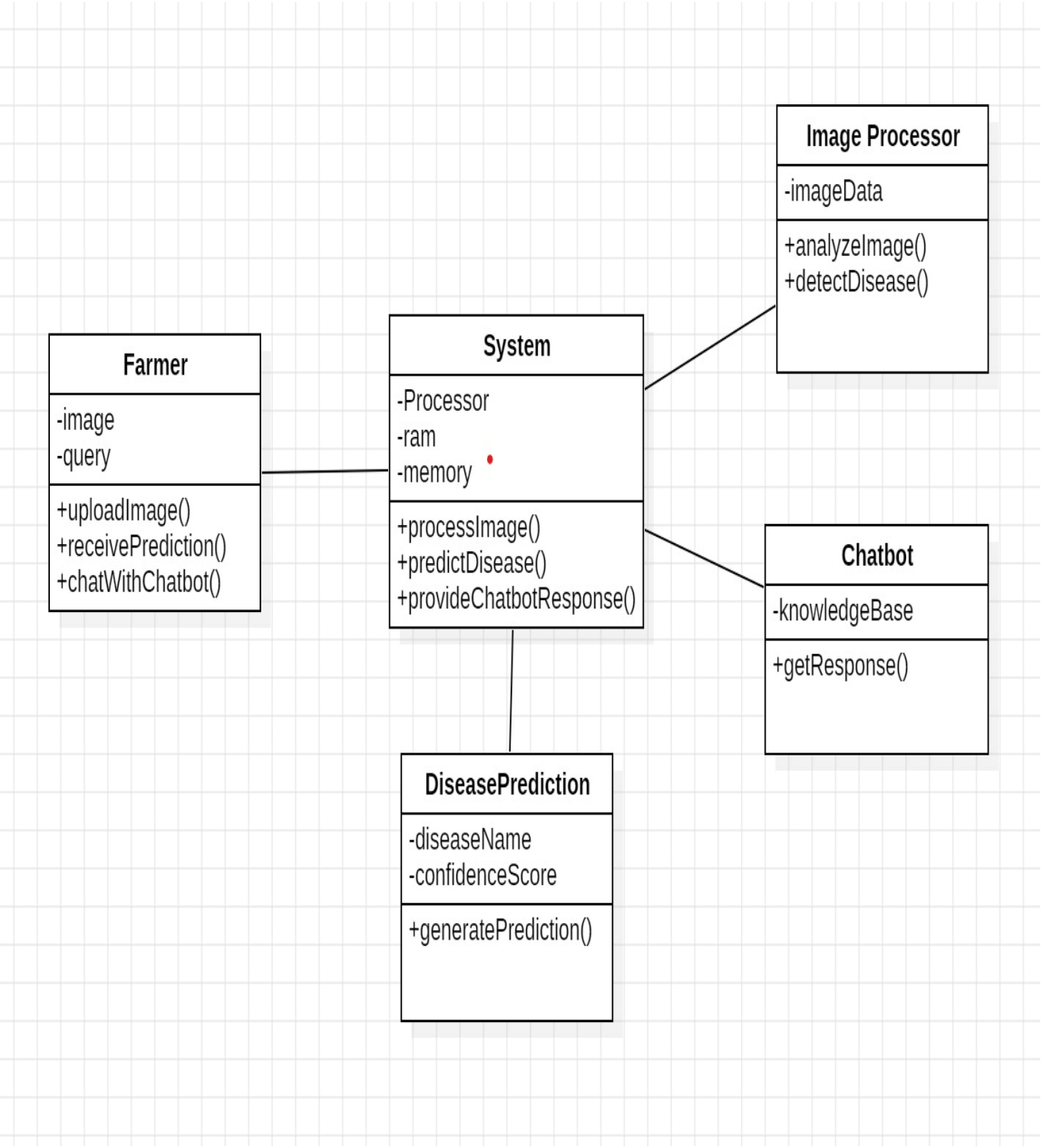


Figure 4.5.1: Class Diagram of the System

4.6 Interaction Diagram:

Sequence Diagram displays the time sequence of the objects participating in the interaction. This consists of the vertical dimension (time) and horizontal dimension (different objects).

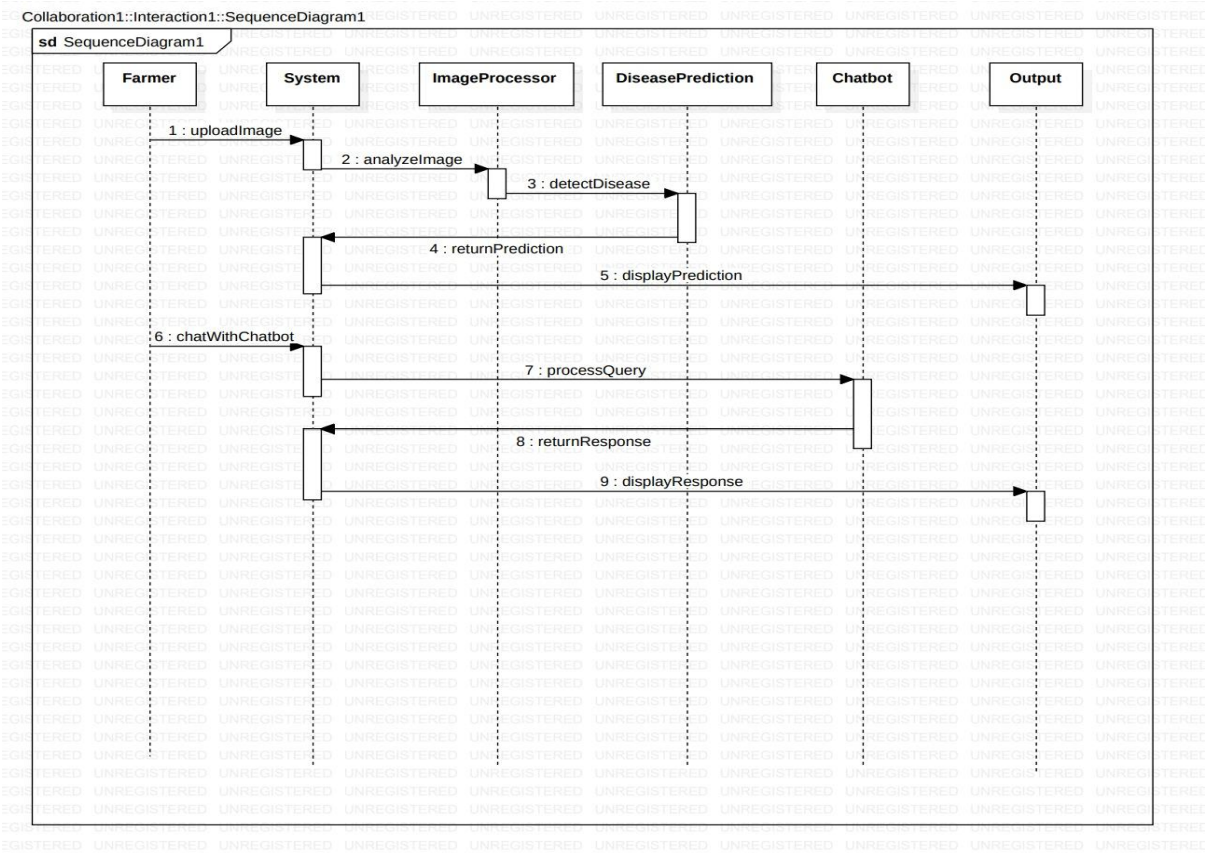


Figure 4.6.1: Sequence diagram of the System

4.7 Activity Diagram

Activity Diagrams describe how activities are coordinated to provide a service which can be at different levels of abstraction. Typically, an event needs to be achieved by some operations, particularly where the operation is intended to achieve a number of different things that require coordination, or how the events in a single use case relate to one another, in particular, use cases where activities may overlap and require coordination. It is also suitable for modelling how a collection of use cases coordinates to represent business workflows.

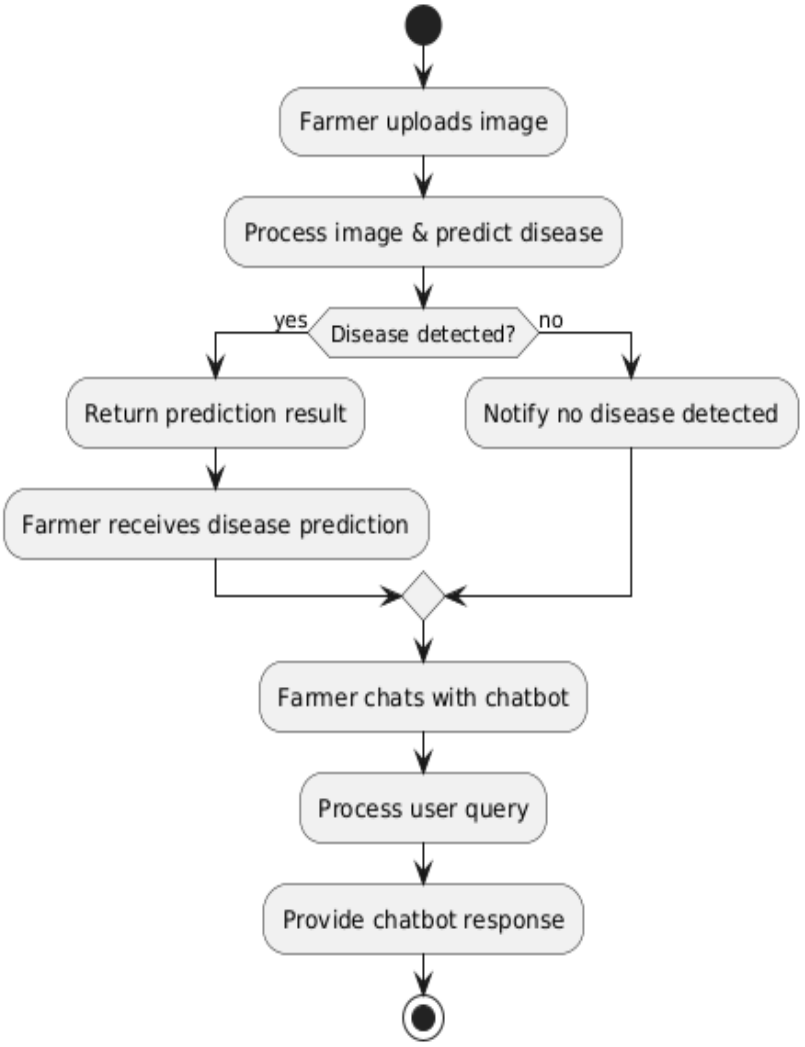


Figure 4.7.1: Activity diagram of the System.

4.8 Summary

After analyzing the interactive diagrams and data flow diagrams of the whole project we have a clear understanding of the procedure that have to be done. We have all the requirements needed for the project and are ready for the implementation phase.

Chapter 5
SYSTEM IMPLEMENTATION

5. SYSTEM IMPLEMENTATION AND SAMPLE CODE

5.1 INTRODUCTION:

System Implementation is the process of integrating the designed system into action or making it functional. It involves developing the system in a way that meets the requirements and objectives specified during the system analysis and design phase. It involves coding the system according to the design specifications. In this chapter all the required libraries are explained briefly.

5.2 PROJECT MODULES:

5.2.1 Flask (Backend Development & API Handling):

Flask is used as the backend framework to create a web server that handles HTTP requests from the frontend. It provides an efficient and lightweight API to process image data uploaded by users and return disease predictions. Flask enables seamless communication between the model and the user interface, ensuring that the system can take inputs, process them, and send back results in real time. The Flask application also manages authentication, request handling, and data storage through different extensions.

Modules Used:

- **Flask:** Acts as the core framework to set up the server and handle HTTP requests.
- **Flask-CORS**: Enables Cross-Origin Resource Sharing (CORS) to allow communication between different domains (e.g., frontend and backend).
- **Flask-JWT-Extended:** Implements secure authentication using JSON Web Tokens (JWT) to restrict unauthorized access.

5.2.2 Deep Learning Model (PyTorch & ResNet-50 for Image Classification):

The disease prediction model is built using a deep learning-based Convolutional Neural Network (CNN). A ResNet-50 model is fine-tuned and trained on a dataset containing images of diseased and healthy crops. The model learns to classify different plant diseases by extracting features from images and mapping them to disease categories. Transfer learning with ResNet-50 helps in improving accuracy with a limited dataset, as the model is already pre-trained on large-scale image data.

Modules Used:

- **torch**: Provides deep learning functionalities, including tensor operations, automatic differentiation, and model training.
- **torchvision.models.resnet50**: Implements a pre-trained **ResNet-50** model, which is fine-tuned for crop disease classification.
- **torch.nn**: Defines neural network layers, activation functions, and loss functions for training the model.
- **torch.optim**: Implements optimization algorithms such as Adam and SGD for efficient learning.
- **torchvision.transforms**: Performs data transformations like resizing, normalization, and augmentation to improve model robustness.
- **torch.utils.data.DataLoader**: Handles batch processing, making training efficient and memory-friendly.

5.2.3 Image Processing & Preprocessing:

Before feeding images into the model, preprocessing is required to standardize input data. Images are resized to match the input size required by **ResNet-50**, normalized to maintain consistent pixel values, and converted into tensors for model compatibility. These transformations enhance model performance by ensuring uniformity across all images.

Modules Used:

- **PIL (Pilllow)**: Handles image loading, resizing, and format conversions.
- **torchvision.transforms**: Applies transformations such as:
    - **Resizing**: Adjusts images to a fixed dimension.
    - **Normalization**: Ensures consistent pixel intensity across images.
    - **Conversion to Tensor**: Transforms images into numerical format suitable for deep learning models.
-

5.2.4 Model Evaluation & Performance Visualization (Matplotlib & Seaborn):

To assess model performance, various evaluation techniques are used. The model's accuracy and loss are tracked over training epochs, and a **confusion matrix** is generated to understand classification errors. Performance metrics help in tuning the model for better accuracy.

Modules Used:

- **Matplotlib**: Visualizes training progress with plots for accuracy and loss.
- **Seaborn**: Creates confusion matrices for evaluating classification performance.
- **sklearn.metrics**: Computes metrics such as precision, recall, and F1-score.
-

5.2.5 Dataset Handling & Management (KaggleHub & OS for Data Access):

With The dataset required for training is sourced from **Kaggle** and handled efficiently to avoid redundancy and maintain structured storage. **KaggleHub** is used to fetch datasets directly from Kaggle, and the **OS** module helps in managing file paths and dataset organization.

Modules Used:

- **KaggleHub**: Downloads datasets directly from Kaggle for training and evaluation
- **OS**: Handles file operations such as creating directories, reading dataset files, and managing image storage paths.

5.3 SAMPLE CODES:

Sample code for ResNet - 50 Implementation:

Importing Required Libraries

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision.transforms as transforms
import torchvision.datasets as datasets
from torch.utils.data import DataLoader, random_split
from torchvision.models import resnet50
```

Data Preprocessing & Augmentation:

```
data_transforms = {
    'train': transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.RandomHorizontalFlip(),
        transforms.RandomRotation(10),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
```

```
    ]),
    'val': transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ])
}
```

Splitting Dataset:

```
train_ratio = 0.8  # 80% training, 20% validation
train_size = int(train_ratio * len(dataset))
valid_size = len(dataset) - train_size

train_dataset, valid_dataset = random_split(dataset, [train_size, valid_size])

dataset_train = DataLoader(train_dataset, batch_size=32, shuffle=True)
dataset_val = DataLoader(valid_dataset, batch_size=32, shuffle=False)
```

Initializing the Model (ResNet50):

```
# Initialize model
model = resnet50(weights="IMAGENET1K_V1")
num_ftrs = model.fc.in_features
model.fc = nn.Linear(num_ftrs, len(dataset_train.classes))  # Adjusting output layer for
    classification
```

Defining Loss Function and Optimizer:

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

Training the model:

```
def train_one_epoch(model, train_loader, criterion, optimizer):
    model.train()
    running_loss = 0.0
    correct = 0
    total = 0
```

```
    for inputs, labels in train_loader:
        inputs, labels = inputs.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        _, predicted = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

    train_acc = 100 * correct / total
    return running_loss / len(train_loader), train_acc

def train_model(model, train_loader, val_loader, criterion, optimizer, num_epochs=10):
    for epoch in range(num_epochs):
        train_loss, train_acc = train_one_epoch(model, train_loader, criterion, optimizer)
        print(f"Epoch [{epoch+1}/{num_epochs}], Loss: {train_loss:.4f}, Train Acc:
    {train_acc:.2f}%")
# Train the model
train_model(model, dataset_train, dataset_val, criterion, optimizer, num_epochs=10)
```

Loading the Trained Model for Predictions:

```
def load_model():
    model = resnet50(weights=None)  # No pretrained weights, since we load our own
    num_ftrs = model.fc.in_features
    model.fc = nn.Linear(num_ftrs, 38)  # Assuming 38 classes, update as needed
    model.load_state_dict(torch.load("plant_disease_model.pth",
    map_location=torch.device("cpu")))
    model.eval()
    return model
from PIL import Image
```

Preprocessing Input Images for Prediction

```
def preprocess_image(image_path):
    transform = transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ])
    image = Image.open(image_path).convert("RGB")
    return transform(image).unsqueeze(0)  # Add batch dimension
```

Predicting on New Images

```
def batch_prediction(test_folder):
    model = load_model()
    predictions = {}
    for img_name in os.listdir(test_folder):
        img_path = os.path.join(test_folder, img_name)
        if img_name.lower().endswith(('.png', '.jpg', '.jpeg')):  # Check if it's an image
            image = preprocess_image(img_path)
            with torch.no_grad():
                output = model(image)
                predicted_class = torch.argmax(output, dim=1).item()
            predictions[img_name] = predicted_class
    return predictions
```

Front End Sample codes

Front End Sample code for Image upload:

```jsx
import React, { useState } from "react";
import "./styles/DiseasePrediction.css";

const DiseasePrediction = () => {
  const [selectedImage, setSelectedImage] = useState(null);
  const [previewImage, setPreviewImage] = useState(null);
  const [prediction, setPrediction] = useState(null);

  const handleImageChange = (event) => {
    const file = event.target.files[0];
    if (file) {
      setSelectedImage(file);
      setPreviewImage(URL.createObjectURL(file));
    }
  };

  const handlePredict = async () => {
    if (!selectedImage) {
      alert("Please upload an image first.");
      return;
    }

    const formData = new FormData();
    formData.append("file", selectedImage);

    try {
      const response = await fetch("http://127.0.0.1:5000/predict", {
        method: "POST",
        body: formData,
      });

      const data = await response.json();
      setPrediction(`Model is predicting it's a ${data.prediction}`);
    } catch (error) {
      console.error("Error predicting:", error);
      setPrediction("Error in prediction");
    }
  };

  return (
    <div className="disease-container">
      <h1>Upload, Analyze, and Heal Your Crops</h1>

      <div className="upload-section">
        <div className="image-container">
          {previewImage ? (
            <img src={previewImage} alt="Uploaded Preview" />
          ) : (
            <img src="/Prediction_img.jpeg" alt="Placeholder" />
          )}
        </div>

        <div className="file-upload">
          <input type="file" accept="image/*" onChange={handleImageChange} className="file-input" />
        </div>
      </div>

      <button onClick={handlePredict} className="predict-btn">
        Predict
      </button>

      {prediction && <div className="prediction-result">{prediction}</div>}
    </div>
  );
};

export default DiseasePrediction;
```

Front End Sample code for interacting with ChatBot

```
  return (
    <div className="planty-container">
      <h1 className="planty-title">🤖🌿 Planty - AI Crop Care Assistant</h1>


      <div className="chat-box">
        <div className="chat-history">
          {chatHistory.map((chat, index) => (
            <div key={index} className={`chat-message ${chat.sender === "User" ? "user-message" : "bot-message"}`}>
              <strong>{chat.sender}:</strong>
              {chat.sender === "Planty" ? (
                <ReactMarkdown>{chat.message}</ReactMarkdown> // Rendering Markdown properly
              ) : (
                <span>{chat.message}</span>
              )}
            </div>
          ))}
        </div>
      </div>

      <div className="input-container">
        <input
          type="text"
          placeholder="Describe the issue with your crop..."
          className="text-input"
          value={inputText}
          onChange={(e) => setInputText(e.target.value)}
        />
        <button onClick={handleSend} className="send-btn">Send</button>
      </div>

      {userChoice === "request_details" && (
        <div className="crop-details">
          <h2>Provide Crop Details</h2>
          <input
            type="text"
            placeholder="Crop Name"
            className="detail-input"
            value={cropDetails.cropName}
            onChange={(e) => setCropDetails({ ...cropDetails, cropName: e.target.value })}
          />
          <input
            type="text"
            placeholder="Symptoms"
            className="detail-input"
            value={cropDetails.symptoms}
            onChange={(e) => setCropDetails({ ...cropDetails, symptoms: e.target.value })}
          />
          <input
            type="text"
            placeholder="Location"
            className="detail-input"
            value={cropDetails.location}
            onChange={(e) => setCropDetails({ ...cropDetails, location: e.target.value })}
          />
          <button onClick={handleCropDetailsSubmit} className="submit-btn">Submit</button>
        </div>
      )}
    </div>
  );
};

export default Planty;
```

Sample Codes for Endpoints

Sample Code for Disease Prediction model Endpoint

```python
@app.route('/predict', methods=['POST'])
def predict():
    if 'file' not in request.files:
        return jsonify({"error": "No file uploaded"}), 400
    file = request.files['file']
    image_path = "temp.jpg"
    file.save(image_path)
    image = preprocess_image(image_path)
    with torch.no_grad():
        output = model(image)
        predicted_class = torch.argmax(output, dim=1).item()
    return jsonify({"prediction": class_names[predicted_class]})
```

Sample Code for Planty Chatbot

```python
@app.route('/predict', methods=['POST'])
def predict():
    if 'file' not in request.files:
        return jsonify({"error": "No file uploaded"}), 400
    file = request.files['file']
    image_path = "temp.jpg"
    file.save(image_path)
    image = preprocess_image(image_path)
    with torch.no_grad():
        output = model(image)
        predicted_class = torch.argmax(output, dim=1).item()
    return jsonify({"prediction": class_names[predicted_class]})
```

Sample Codes for API

Sample Code for Interacting with Gemini API

```
def get_gemini_response(question):
    try:
        if question.lower() in greetings:
            return random.choice(["Hello! How can I assist you with plant diseases?", "Hi there! Need help
            diagnosing a plant issue?", "Hey! What plant-related query do you have?"])

        if not is_plant_related(question):
            return "I'm designed to assist only with plant diseases and related topics. Please ask me
            about plants!"

        response = chat.send_message(question, stream=True)
        bot_reply = ''.join([chunk.text for chunk in response])

        keywords = ["fungal", "bacterial", "viral", "nutrient deficiency", "pesticide", "climate"]
        follow_up_questions = {
            "fungal": "Would you like to know about effective antifungal treatments?",
            "bacterial": "Should I provide natural remedies or chemical treatments?",
            "viral": "Would you like guidance on preventing viral spread in crops?",
            "nutrient deficiency": "Do you need recommendations for fertilizers or soil amendments?",
            "pesticide": "Are you looking for eco-friendly pesticide options?",
            "climate": "Do you want insights on how weather affects plant diseases?"
        }

        for key in keywords:
            if key in bot_reply.lower():
                return f"{bot_reply}\n\n{follow_up_questions[key]}"

        return bot_reply
    except Exception as e:
        return f"Error: {str(e)}"
```

5.4 SUMMARY:

Hence, this chapter clearly explains all the implementation steps required to do this project. We have implemented the model, now we have to test the model under various testcases.

Chapter 6
SYSTEM TESTING

6 TESTING

6.1 INTRODUCTION

Testing is carried out on a fully developed system to identify whether the system works according to the requirements specified and also to identify and correct the errors. It can be either done manually or using automated tools.

6.1.1. Goals of Software Testing

- **Verification:** It allows testers to confirm that the software meets the various business and technical requirements specified before the development of the project.
- **Validation**: Confirms that the software performs as expected and as per requirements of the clients after the development of the project. It involves comparing the final output with the expected output and then making necessary changes if there is a difference.
- **Defects**: The important purpose of testing is to find different defects in the software to prevent its failure or crash during implementation. Defects if left undetected or unattended can harm the functioning of the software.
- **Transparency**: With the assistance of reports generated during the process of software testing, testers can accumulate a variety of information related to the software and the steps taken to prevent failure.
- **Quality Analysis**: Testing helps improve the quality of software by constantly verifying design, code and outcome.
- **Compatibility**: It helps to validate application's compatibility with the implementation environment, various devices, operating systems, user requirements among other things.
- **Performance**: It helps to ensure that system performs efficiently by testing under different scenarios.

6.2 Testing Methods

- **Static testing**: It is also known as Verification in Software testing. Verification is the processs carried out before the development of the project to verify whether the project is being developed according to the requirements or not.
- **Dynamic testing**: It is also known as Validation in Software testing. Validation is the process carried out after the development of the project to confirm that the project is developed according to requirements or not.

6.3 Testing Approaches

There are three types of testing approaches

- **White Box testing**: It is also called as Glass Box/Clear Box/Structural testing. White Box testing is based on application's internal code structure. In White Box testing, an internal perspective of the system, as well as programming skills is used to design test.
- **Grey Box testing**: Grey Box is the combination of both White Box and Black Box Testing. The tester who works on this type of testing needs to have access to design documents. This helps to create better test cases in this process.
- **Black Box testing**: It is also called as Behavioral / Specification-Based/Input Output Test It is a software testing method in which testers evaluate the functionality of the software under test without looking at the internal code structure.

6.3.1 Testing Levels

- **Unit testing**: It is done to check whether the individual modules of the source code are working properly i.e., testing each and every unit of the application separately by the developer in the developer's environment.
- **Integration testing**: Integration testing is the process of testing the connectivity or data transfer between a couple of units tested modules. It is also called String testing. It is sub divided into Top-Down approach, Bottom- Up approach and Sandwich approach.
- **System testing**: It is also called end to end testing. It is Black Box testing. Testing the fully integrated application this is also called as end to end scenario testing. Verify testing of every input in the application to check for desired outputs
- **Acceptance testing**: To obtain customer sign-off so that software can be delivered and payments received. Types of Acceptance testing are Alpha, Beta and gamma testing.

6.4 Testing Strategy

The testing strategy employed is **Functional testing**.Functional testing is a type of Software testing that validates the software system against the functional requirements/specifications and tests each function of the software application, by providing appropriate input, verifying the output against the Functional requirements.Functional testing is one of the types of System testing.

6.5 Sample Test Cases

Testcase 1: Unit Testing in Image Module

Input: Uploading a valid Image

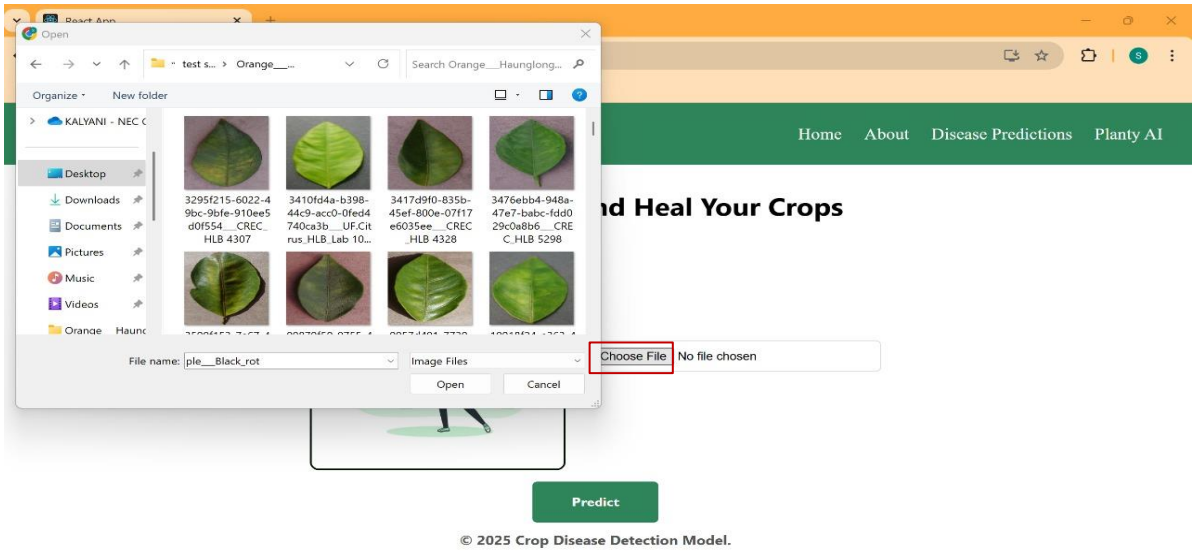


Figure 6.5.1: Uploading image

Expected Output: Image should be uploaded and should be visible on the screen

Observed Output:

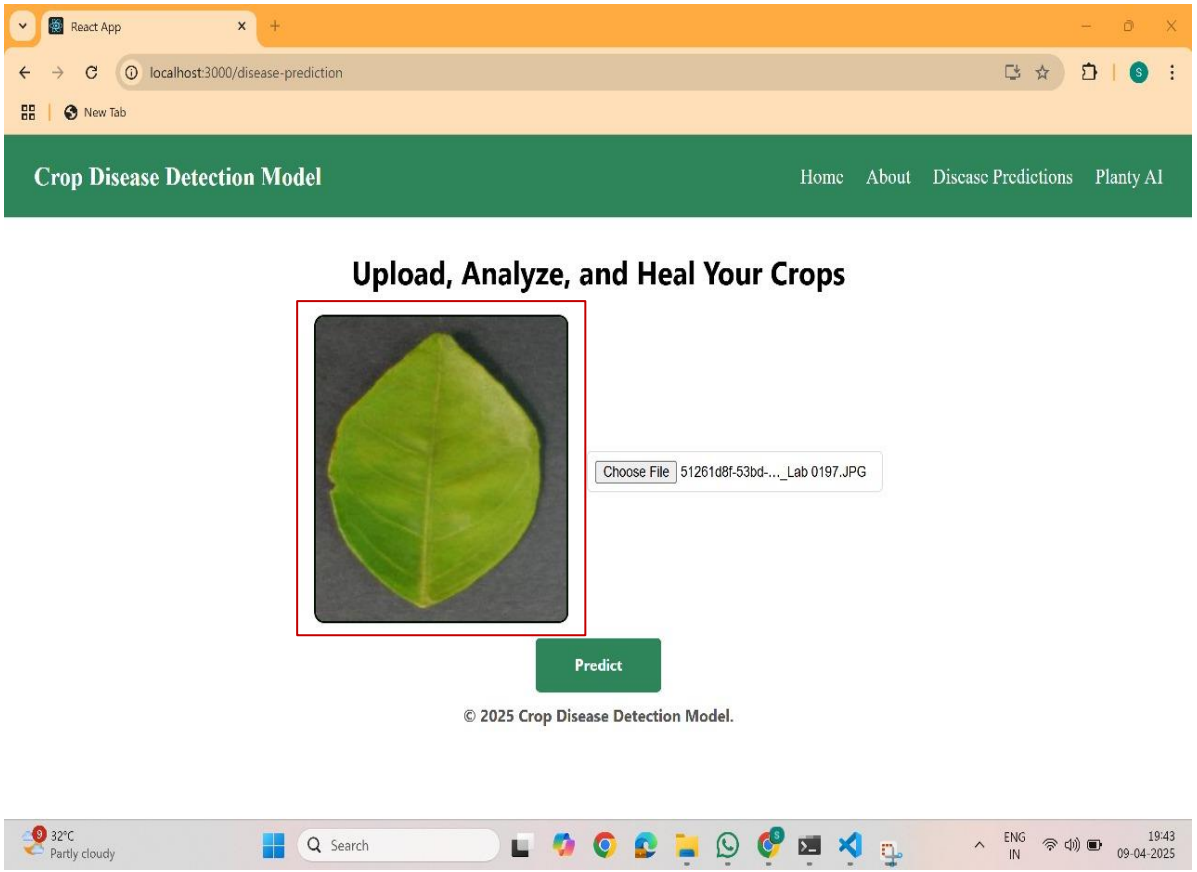


Figure 6.5.2: Uploaded image is visible on the screen

Testcase 2: Validation Testing in Image Module

Input: No image is selected

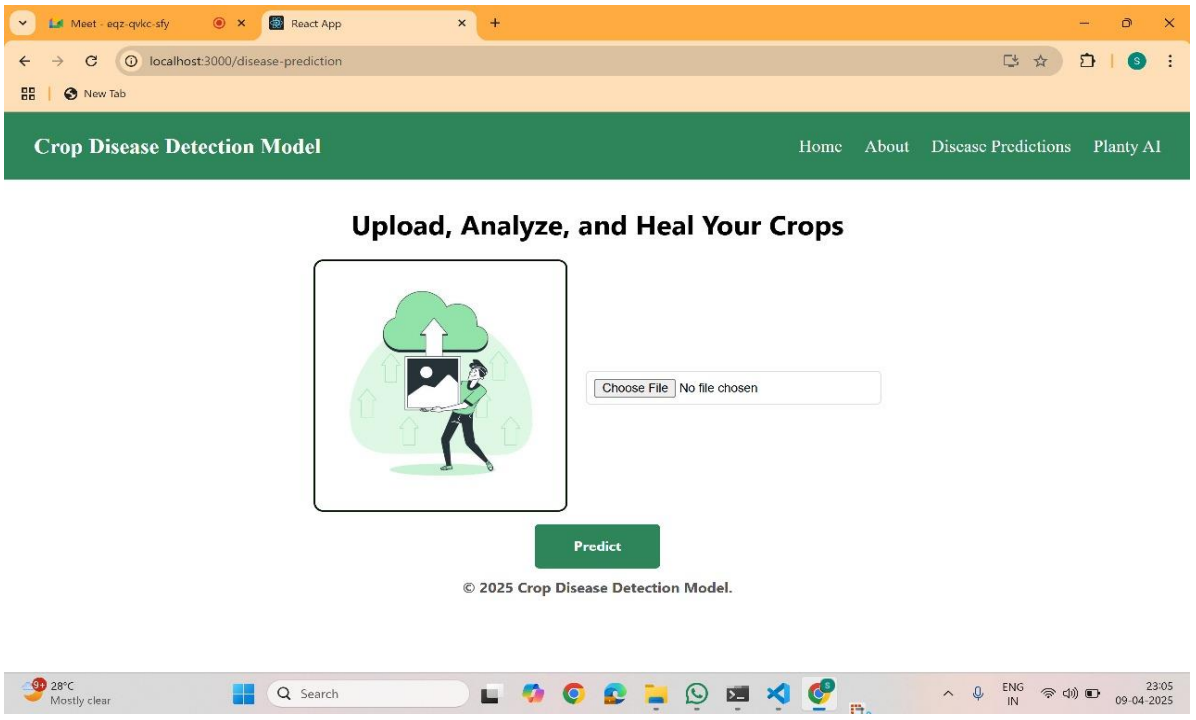


Figure 6.5.3: Interface without image input

Expected Output: A pop up is shown that to please upload an image first

Observed Output:

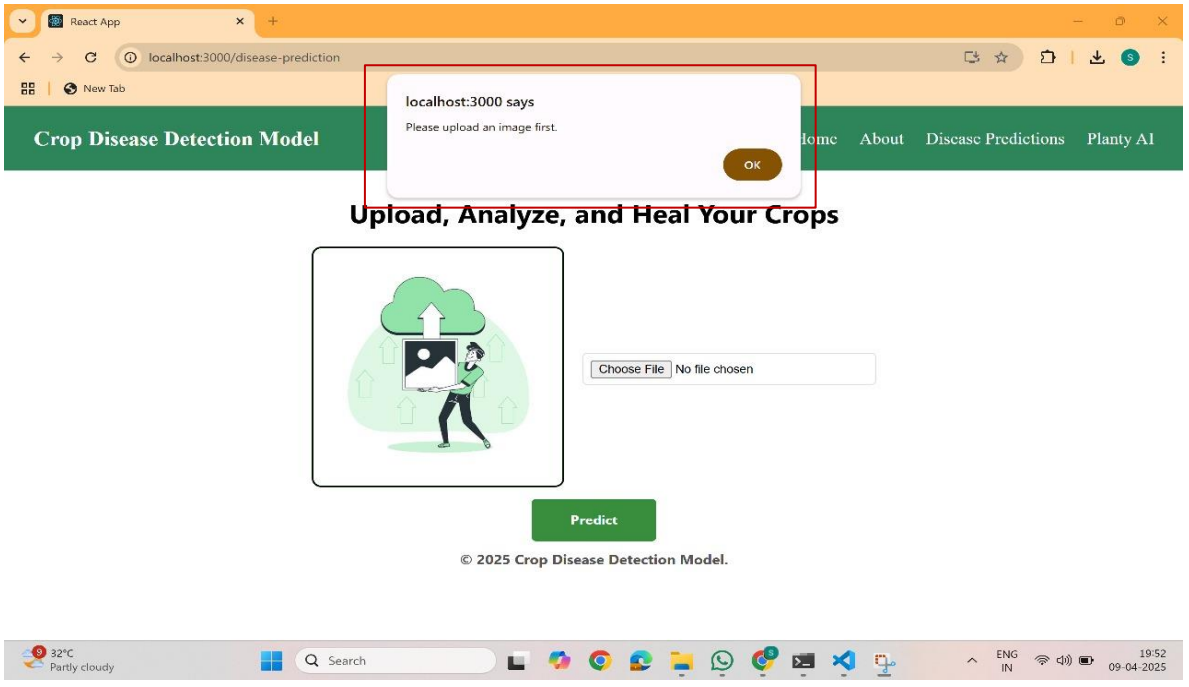


Figure 6.5.4: Input upload request

Testcase 3: Integration testing in crop disease detection Module

Input: Uploading a leaf image for prediction

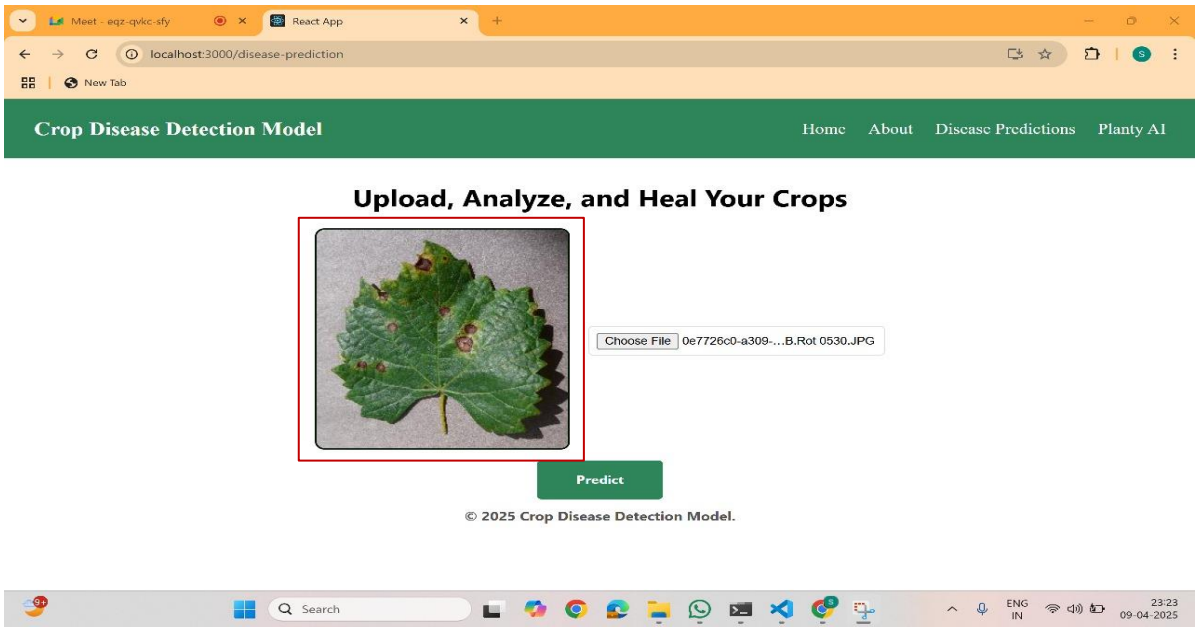


Figure 6.5.5: Uploaded image

Expected Output: System should predict the disease and show the disease

Observed Output:

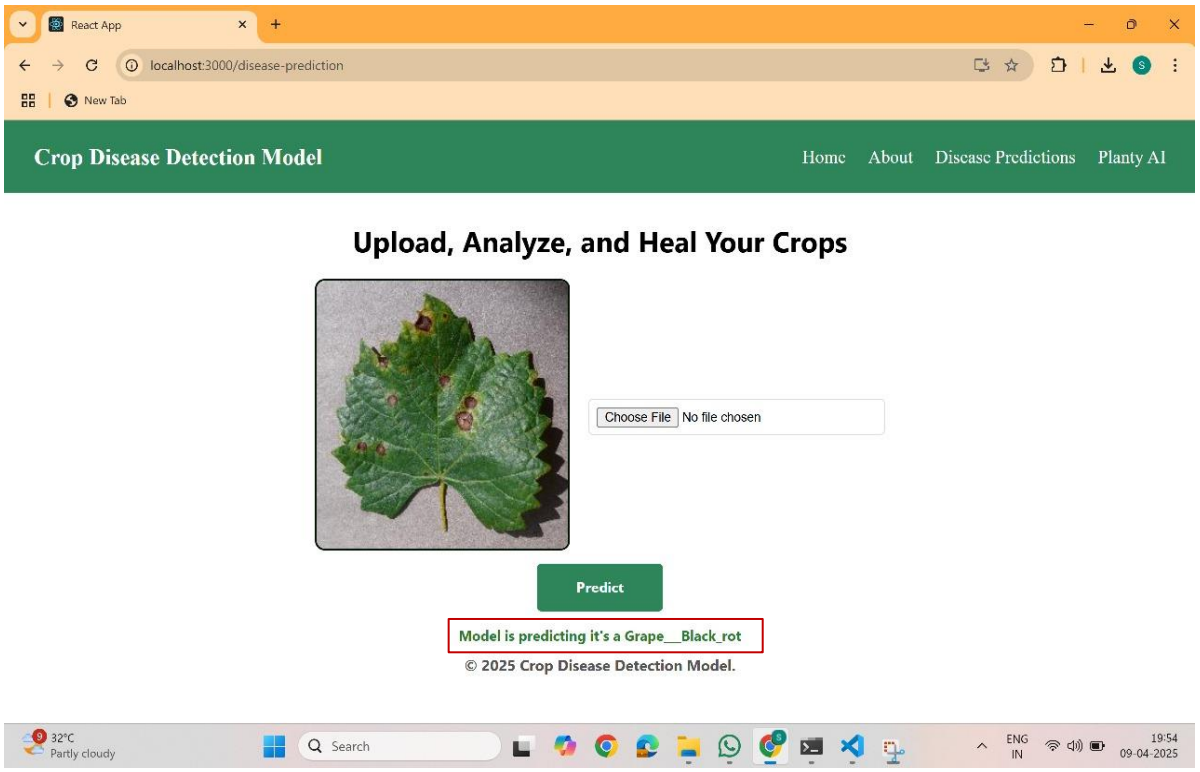


Figure 6.5.6: System predicts the leaf disease correctly

Testcase 4: Validation testing in crop disease detection module

Input: Uploading an invalid image (non - leaf image)

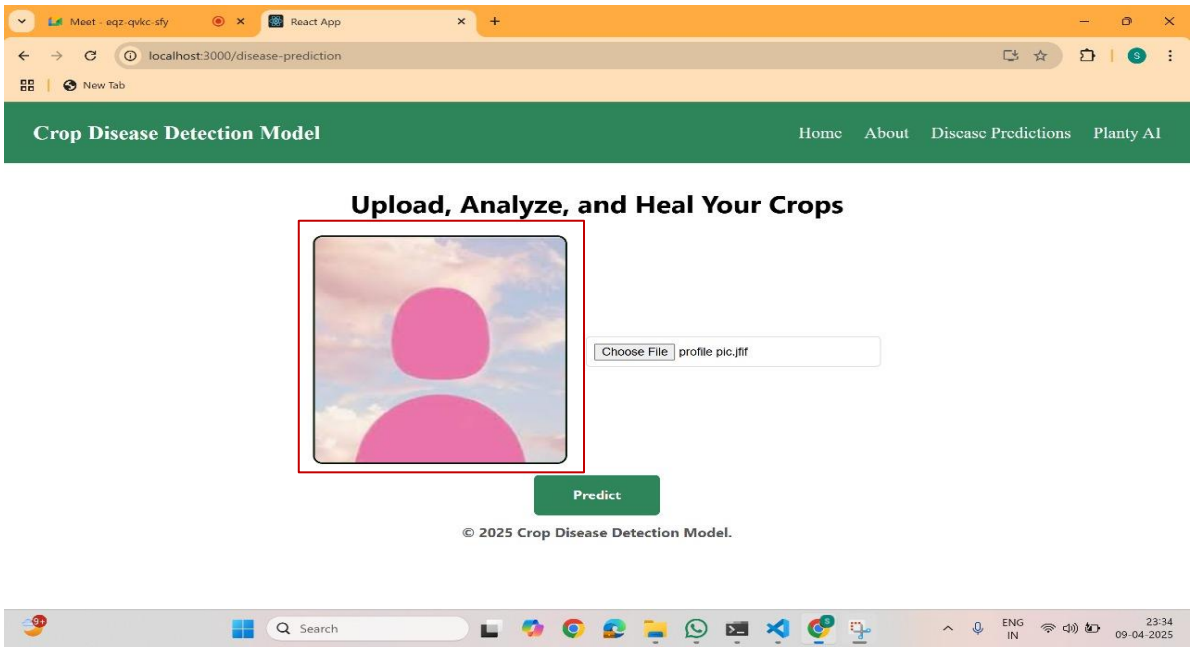


Figure 6.5.7: Non leaf image uploaded

Expected Output: System should show it is undefined

Observed Output:

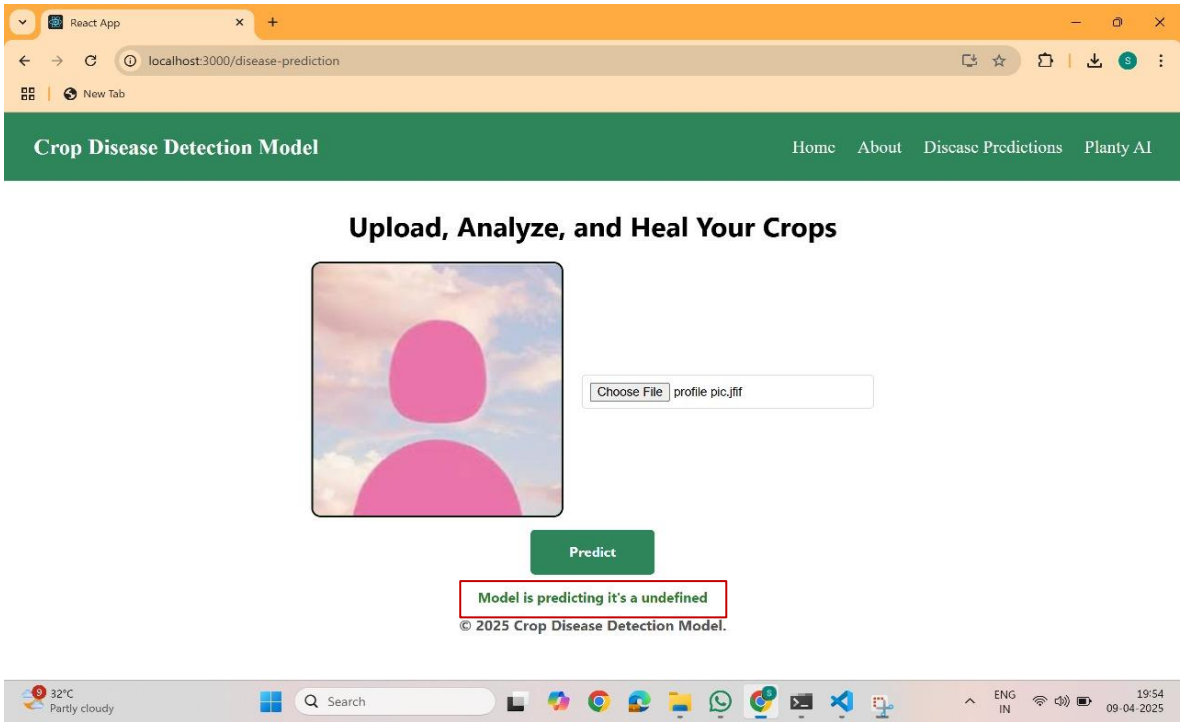


Figure 6.5.8: System predicted undefined

Testcase 5: Integration testing in planty AI module

Input: Giving a valid prompt

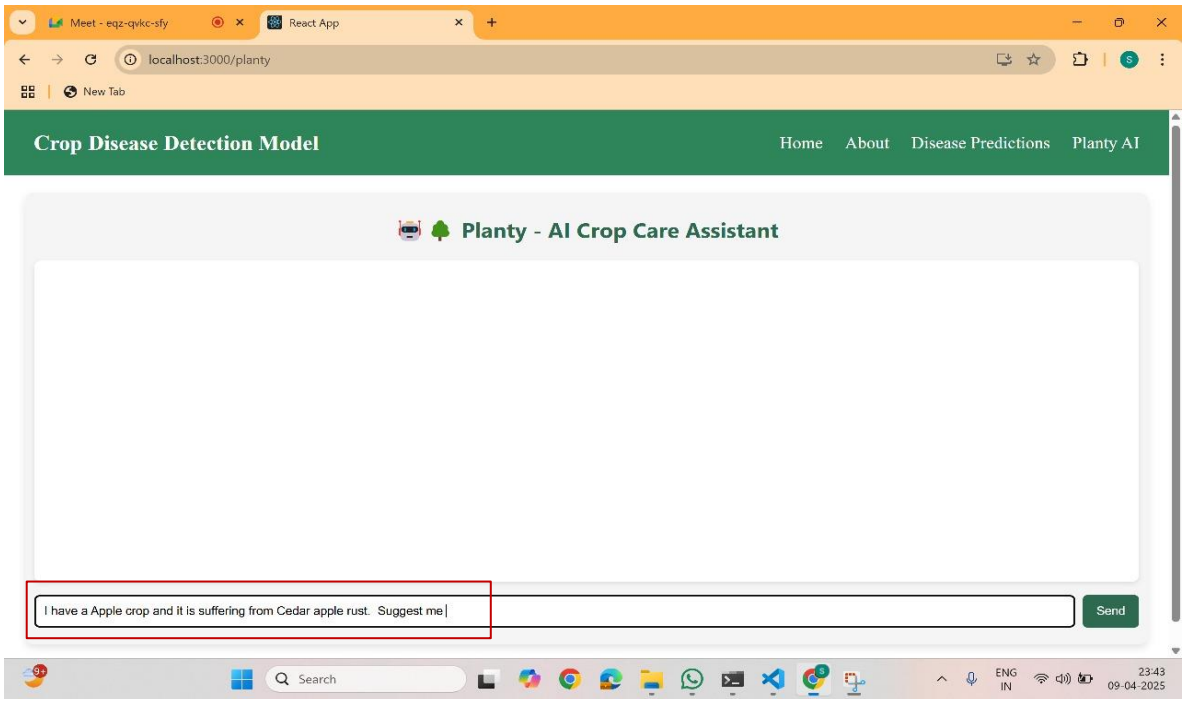


Figure 6.5.9: Giving a valid prompt

Expected Output: Chatbot should reply with info and suggestions related to the disease

Observed Output:

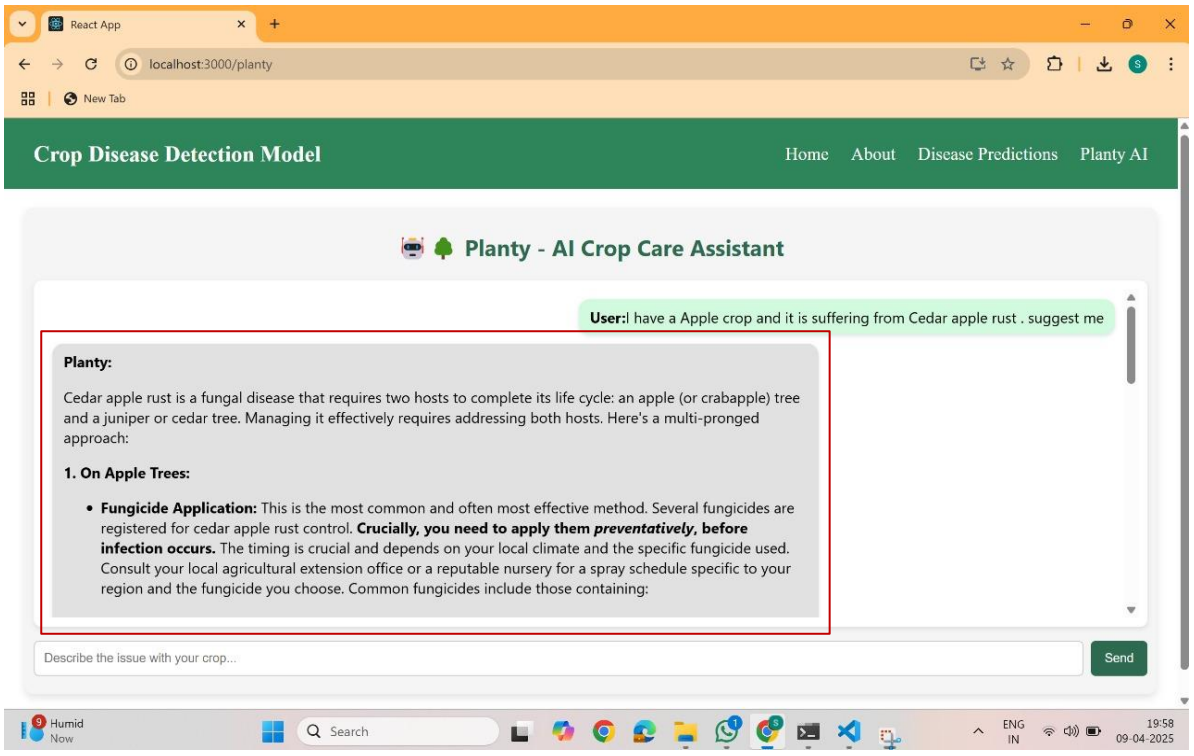


Figure 6.5.10: Related response for the query is observed

Testcase 6: Validation testing in planty AI module

Input: Giving an unrelated prompt

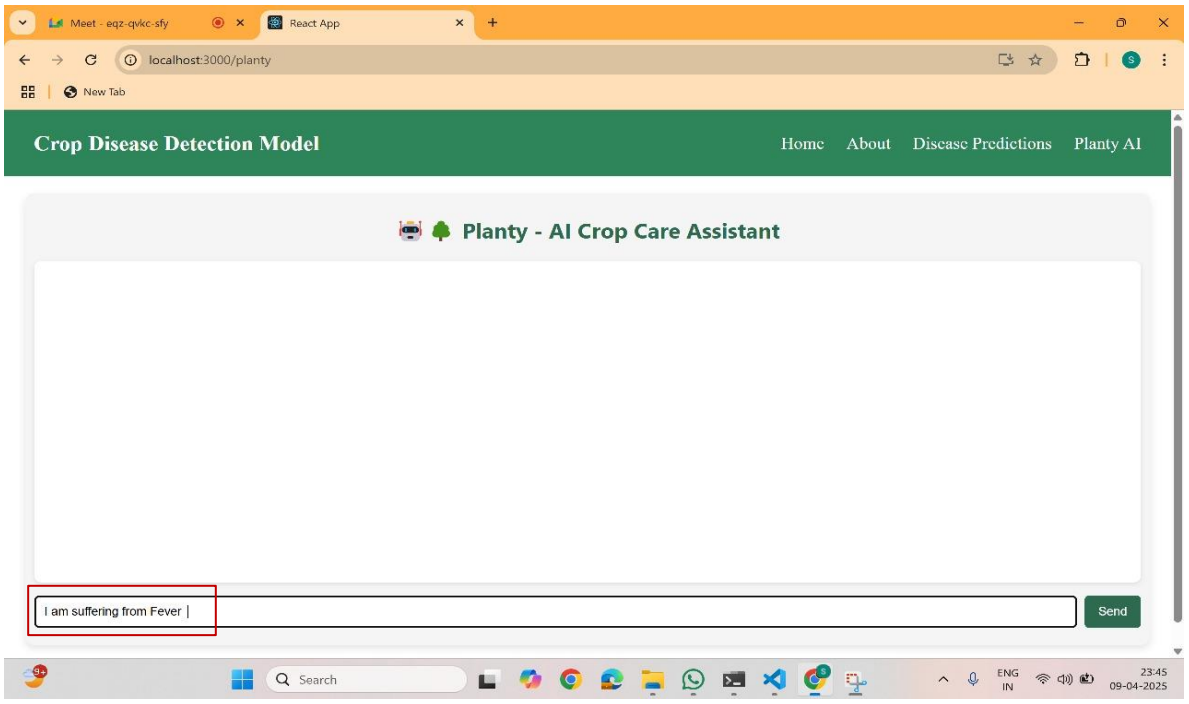


Figure 6.5.11: Giving an unrelated prompt

Expected Output: Chatbot should remind users with a response that it can't be used for general chatting

Observed Output:

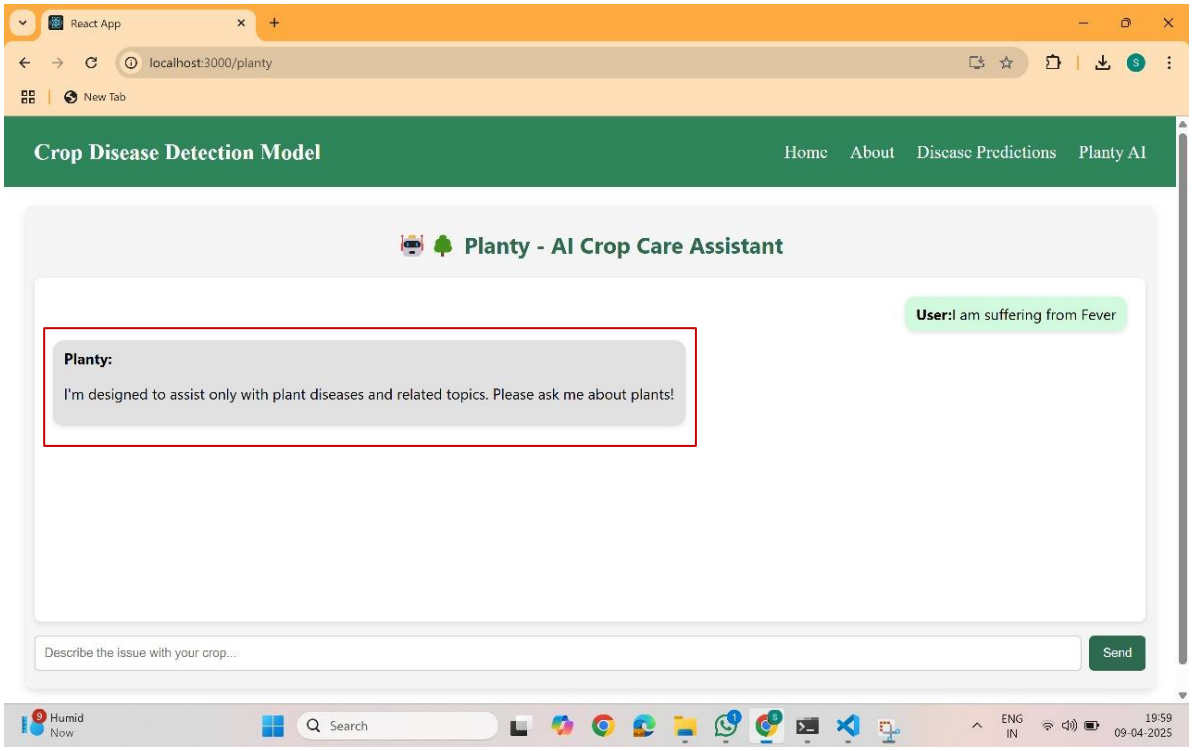


Figure 6.5.12: Related response is observed from the ChatBot

6.6 SUMMARY

In this chapter, we have listed out different test cases. we have tested the model with using Convolutional Neural Networks. All the test cases were successfully passed using CNN.

Chapter 7
RESULTS

7. RESULTS

System results are the outcome of the whole process of software testing life cycle. The results thus produced, offer an insight into the deliverables of a software project, significant in representing the status of the project to the stake holders. The analysis of system results is an important part of the system evaluation, as it provides insight into the effectiveness and efficiency of the system and helps to identify areas for improvement.

7.1 User Interface

1. Initially Our Home Page will look like this

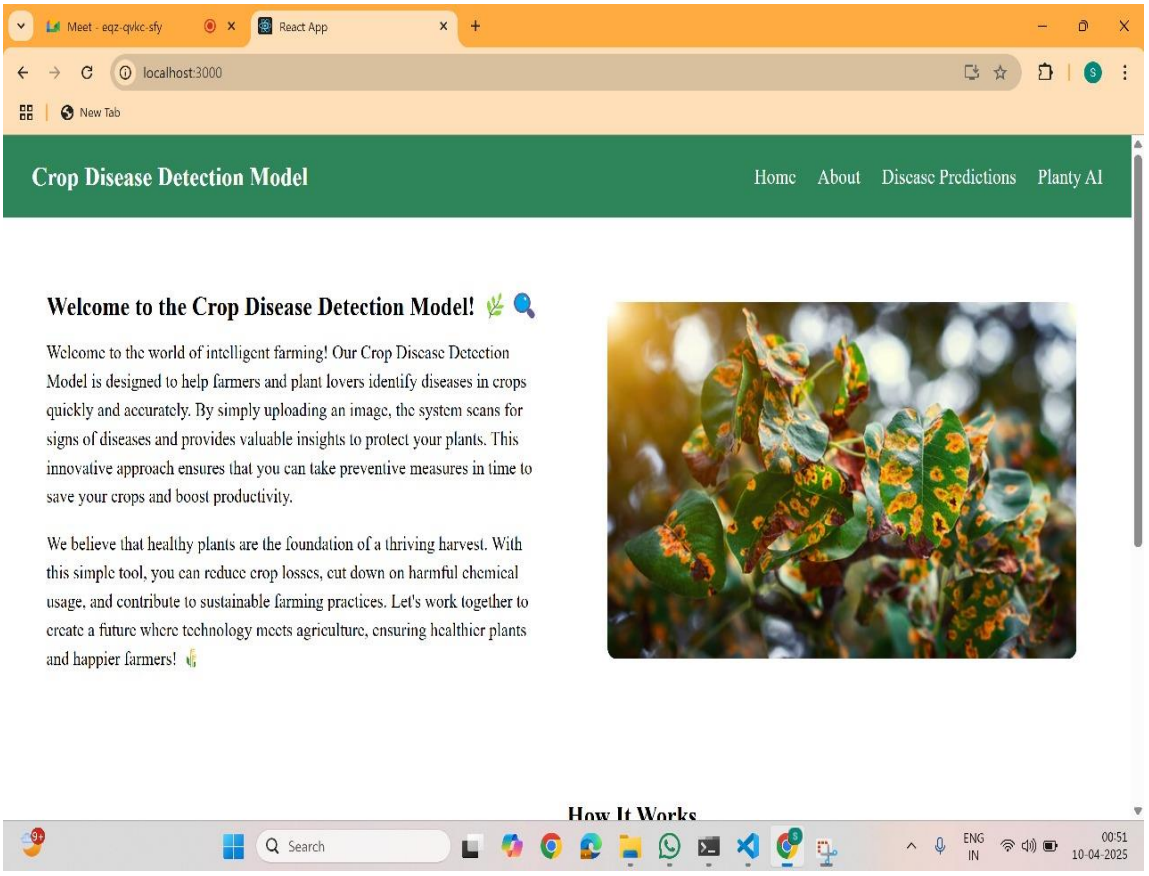


Figure 7.1.1: Home Page Interface

2. When the user clicks on the Disease Predictions option then the user interface for the Disease Predictions will look as follows.

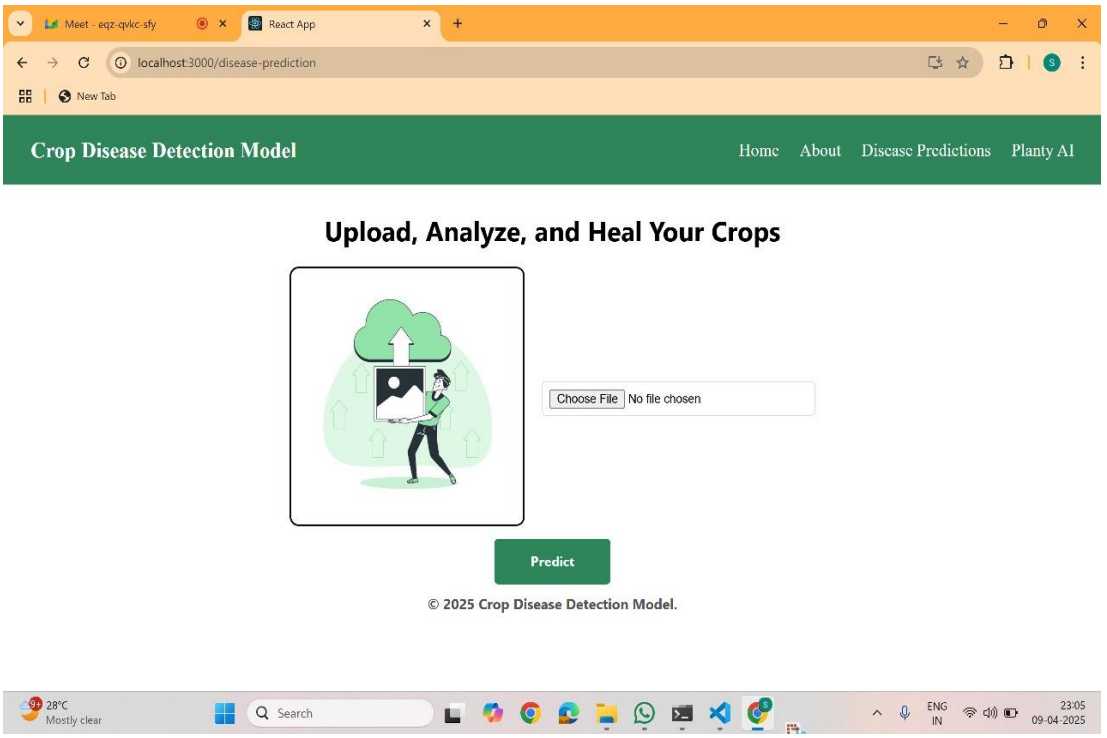


Figure 7.1.2**:** Disease Predictions Interface

3. When the user clicks on the Planty AI option then the user interface for the Planty AI will look as follows.

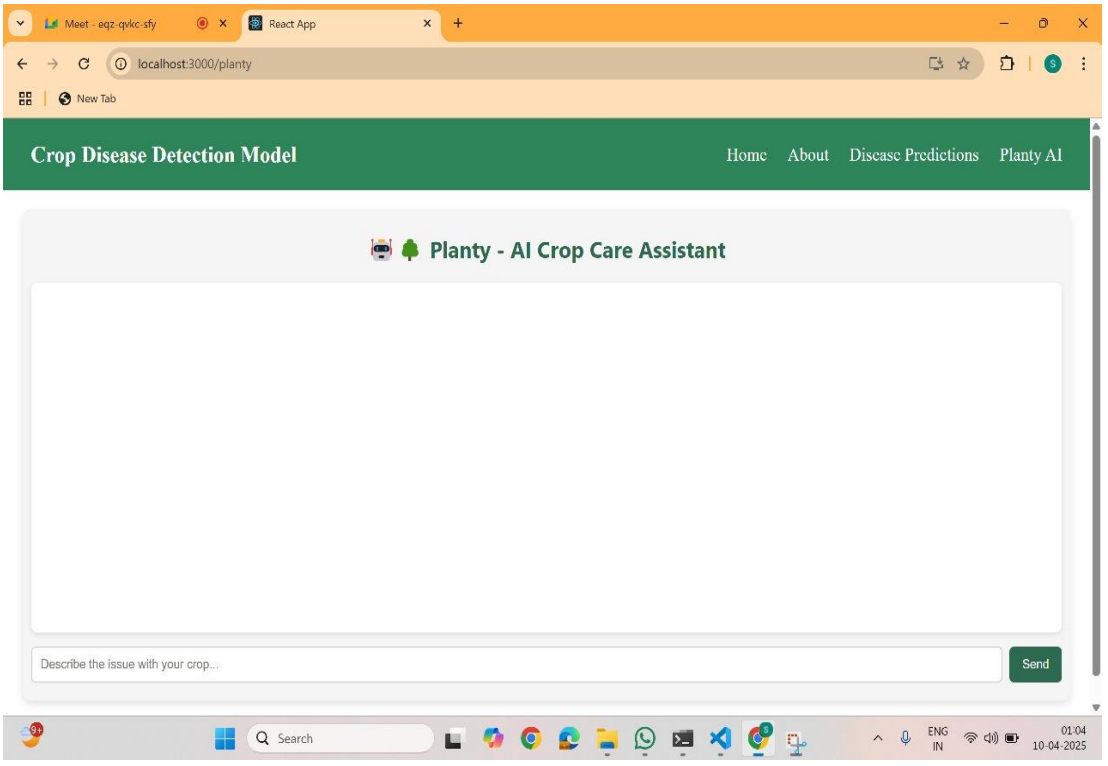


Figure 7.1.3**:** Planty AI Interface

7.2 Upload Image

1. The user can upload the image in the disease prediction page.

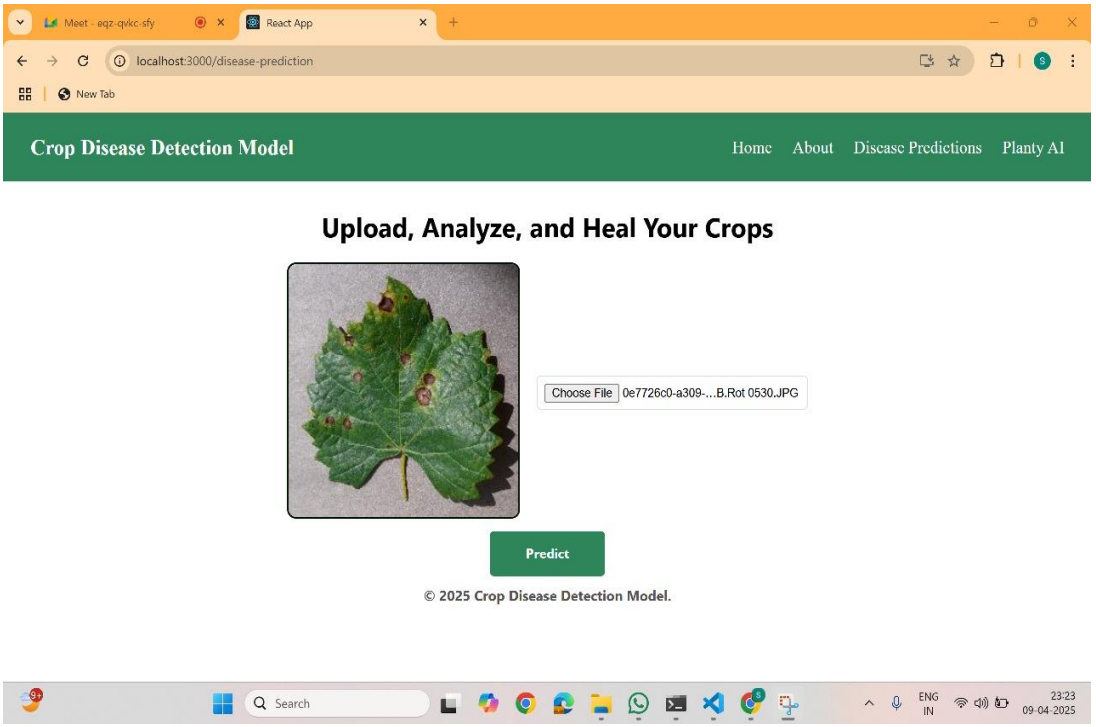


Figure 7.2.1: Image uploaded

2. After uploading the image, the user can click on the predict button and know the type of the disease

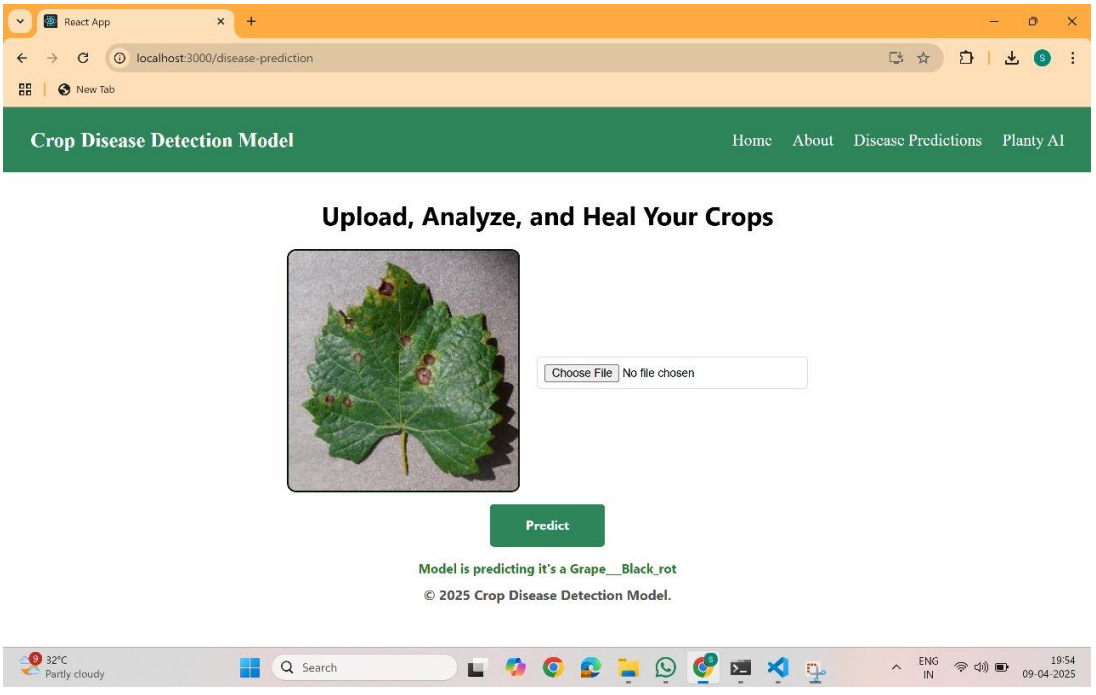


Figure 7.2.2: Predicting the disease

7.3 Planty AI

1. Here, user can give the queries related to crop diseases

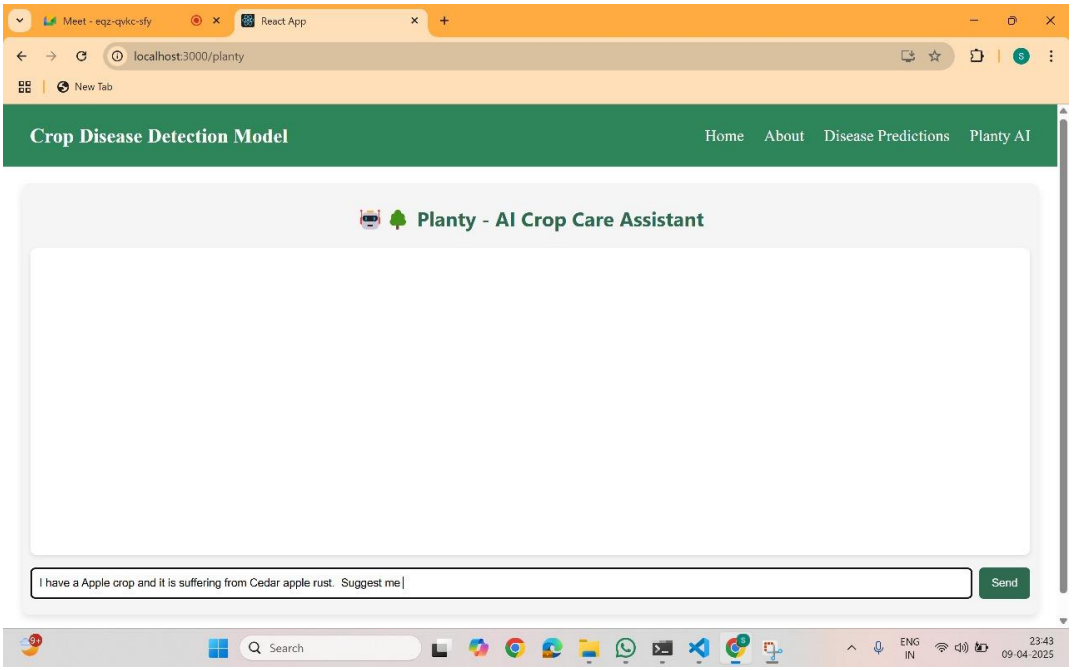


Figure 7.3.1**:** Prompting a query

2. When the user clicks on the send button, they receive relevant suggestions related to the detected disease.

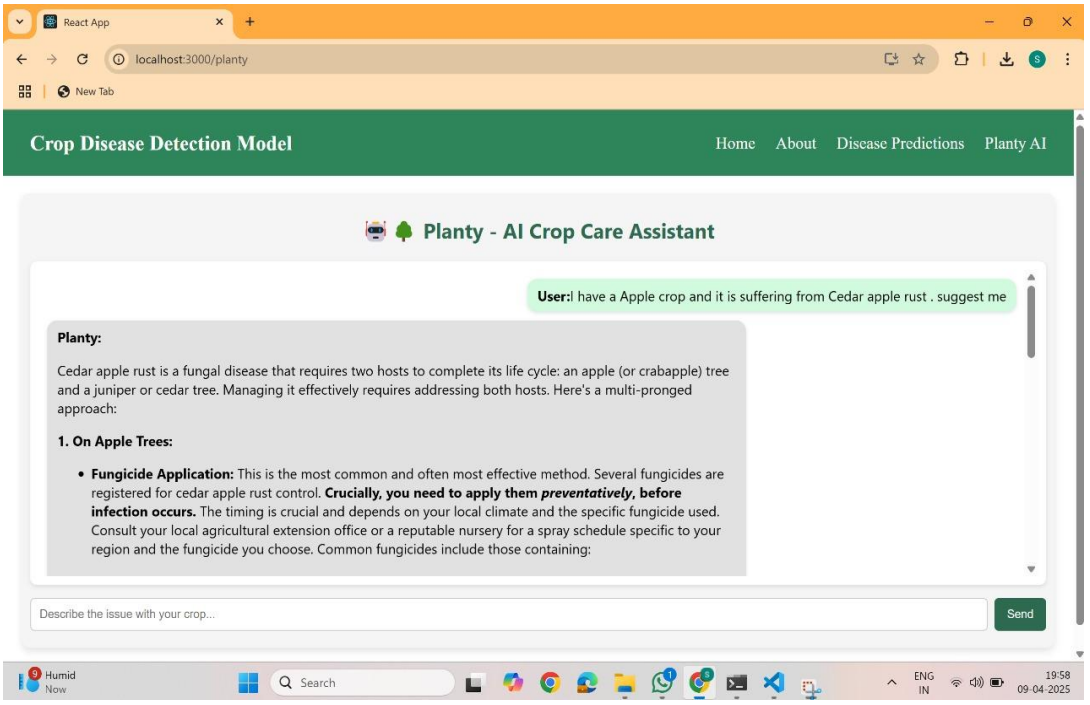


Fig 7.3.2**:** Chatbot response for related query

7.1 SUMMARY

In this chapter, we have listed out different results of our application. We have listed all the possible results that are expected from our application.

Chapter 8
CONCLUSION AND FUTURE WORK

CONCLUSION

The proposed crop disease detection using ResNet-50 successfully integrates deep learning with AI-powered chatbot assistance to support farmers in diagnosing plant diseases and providing relevant remedies. By leveraging ResNet-50, the system effectively classifies plant diseases with high accuracy 99.04%, ensuring quick and reliable results. The Gemini API-powered chatbot further enhances usability by offering guidance and recommendations based on user queries. This system is designed with a Flask-based backend that seamlessly connects the React.js frontend with the deep learning model and chatbot API. The use of Google Colab and embedded GPUs optimizes model execution. Overall, the project demonstrates a scalable and accessible approach to agricultural disease diagnosis, helping farmers make informed decisions and improve crop health.

FUTURE WORK

Future enhancements to the system include real-time detection through mobile applications or IoT-enabled cameras, enabling farmers to instantly scan plants for disease identification and receive immediate recommendations, leading to quicker decision-making and improved crop health. Implementing on-device inference would allow offline functionality, ensuring accessibility for farmers in remote areas with limited internet connectivity. To enhance inclusivity, multilingual chatbot support can be introduced, enabling users to interact in regional languages for better understanding of disease remedies and guidance. Additionally, a user feedback mechanism can be integrated, allowing farmers to validate disease predictions and chatbot responses, continuously improving model accuracy and ensuring more reliable recommendations based on real-world interactions.

Chapter 9
REFERENCES

REFERENCES

[1] https://www.python.org

[2] https://ieeexplore.ieee.org/abstract/document/10895694

[3] https://ieeexplore.ieee.org/document/10183541

[4] https://arxiv.org/abs/1512.03385

[5] https://ieeexplore.ieee.org/abstract/document/9068805

[6] https://pytorch.org/vision/main/models/generated/torchvision.models.resnet50.html

[7] https://ieeexplore.ieee.org/document/9067661

[8] https://ieeexplore.ieee.org/document/8993089

[9] https://ai.google.dev/gemini-api/docs

[10] https://flask.palletsprojects.com/en/stable/

Chapter 10

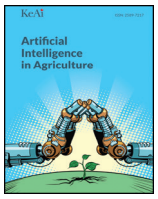
BASE PAPER

Artificial Intelligence in Agriculture 6 (2022) 23–33

Contents lists available at ScienceDirect

Artificial Intelligence in Agriculture

journal homepage: http://www.keaipublishing.com/en/journals/artificial-intelligence-in-agriculture/



Transfer Learning for Multi-Crop Leaf Disease Image Classification using Convolutional Neural Network VGG

Ananda S. Paymode *, Vandana B. Malode

MGM's Jawaharlal Nehru Engineering College, Aurangabad 431001, Maharashtra, India

ARTICLE INFO

Article history:
Received 9 October 2021
Received in revised form 8 December 2021
Accepted 30 December 2021
Available online 7 January 2022

Keywords:
Convolutional Neural Network (CNN)
Artificial Intelligence (AI)
Visual Geometry Group (VGG)
Multi-Crops Leaf Disease (MCLD)

ABSTRACT

In recent times, the use of artificial intelligence (AI) in agriculture has become the most important. The technology adoption in agriculture if creatively approached. Controlling on the diseased leaves during the growing stages of crops is a crucial step. The disease detection, classification, and analysis of diseased leaves at an early stage, as well as possible solutions, are always helpful in agricultural progress. The disease detection and classification of different crops, especially tomatoes and grapes, is a major emphasis of our proposed research. The important objective is to forecast the sort of illness that would affect grapes and tomato leaves at an early stage. The Convolutional Neural Network (CNN) methods are used for detecting Multi-Crops Leaf Disease (MCLD). The features extraction of images using a deep learning-based model classified the sick and healthy leaves. The CNN based Visual Geometry Group (VGG) model is used for improved performance measures. The crops leaves images dataset is considered for training and testing the model. The performance measure parameters, i.e., accuracy, sensitivity, specificity precision, recall and F1-score were calculated and monitored. The main objective of research with the proposed model is to make on-going improvements in the performance. The designed model classifies disease-affected leaves with greater accuracy. In the experiment proposed research has achieved an accuracy of 98.40% of grapes and 95.71% of tomatoes. The proposed research directly supports increasing food production in agriculture.

1. Introduction

To contribute to the development of nations, knowledge of agriculture sectors is crucial. Agriculture is a one-of-a-kind source of wealth that develops farmers. For a strong country, the development of farming is a necessity and a need in the global market. The world's population is growing at an exponential rate, necessitating massive food production in the next 50 years. Information about different types of crops and diseases occurring at each level and its analysis at an early stage play a key and dynamic role in the agriculture sector. A farmer's main problem is the occurrence of various diseases on their crops. The disease classification and analysis of illnesses is a crucial concern for agriculture's optimum food yield. Food safety is a huge issue due to a lack of infrastructure and technology, so crop disease classification and identification are important to be considered in the coming days. This is necessary for yield estimation, food security, and disease management. Detection and recognition of crops illnesses is an important study topic because it could be capable of monitoring huge fields of crops and detecting disease symptoms as soon as they occur on plant leaves. As a result, finding a quick, efficient, least inexpensive, and effective approach to determine crops diseases instances is quite important (C. J. Chen et al., 2021).

Artificial intelligence (AI) provides considerable assistance to agriculture, which enhances a nation's gross domestic product (GDP) mostly through this sector. Climate change, labour scarcity, rainy season uncertainty, natural disasters, and various diseases on plant leaves are all major issues in agriculture. The plant leaves recognition and detection studies with edge intelligence applied to agriculture. There is a new advancement with different deep learning models that overcomes the challenge. The YOLOv3 neural network model is based on deep learning and is built on an embedded system and the NVIDIA Jetson TX2. The system is implemented on a drone, and photographs of plants are taken, pest positions are identified, and pesticides are applied as needed; this is a novel approach based on deep learning (Al Hiary et al., 2011).

Hyper spectral and multispectral knowledge acquisition techniques and applications have exhibited their utility in improving agricultural production and practises by providing farmers and agricultural management with crucial data on the elements impacting crop condition and growth. This technology has been widely employed in a variety of agricultural applications, including sustainable agriculture (Ang, 2021).

* Corresponding author.
E-mail addresses: anandpaymode@gmail.com (A.S. Paymode), vandanamalode@jnec.ac.in (V.B. Malode).

https://doi.org/10.1016/j.aiia.2021.12.002

Weed detection in vegetable plantations is more difficult than in crop plantations due to uneven plant spacing. Deep Learning technology is a novel method that blends with image processing. This approach concentrates solely on recognising plants, avoiding the handling of numerous plant species. Furthermore, by reducing the amount of training image collection and even the complexity of weed detection, this technique can improve plant diagnosis accuracy and performance (Jin et al., 2021).

The massive crop loss occurred because of the failure to predict disease at an early stage, which always results in lower crop production. As a result, identifying and analysing crop diseases is a critical step in ensuring crop quality (Wu, 2020). As high computing speed and power have recently improved, the availability of massive datasets improves the system's efficiency.

In this section, there are various techniques for detecting and classifying crop leaf disease. We present the related survey as a system that employs a variety of classifier techniques. There are two types of combinations: serial and hybrid, with the combination of serial and parallel achieving the significant performance parameter within 600 images (Massi et al., 2020). The hybrid combination has a recognition rate of 91.11%, which is higher than the serial, parallel, and deep learning approaches. For identifying and analyzing leaf illness, a deep learning convolutional neural network (CNN) model was used to classify healthy and sick images. The model train contained 25 different plants, 58 classes' sets, including healthy and diseased plants, and had 87,848 images. Using several (Ferentinos, 2018) model architectures, the best performance success rate was 97.53%. The Multi-Context Fusion Network (MCFN), a deep learning-based method, is built and prepared for crop disease detection. The MCFN aids in the extraction of visual information from 50,000 crop photos. The MCFN produced 77 common crops infected using a deep fusion model, with a 97.50% identification accuracy (Jin et al., 2021).

The identification of weeds in crops using the CovNet algorithm is also a potent and cutting-edge approach. In recent research, bounding boxes were drawn across cropped images and the model was trained. Colour-based segmentations are applied to images and colour information, and visual categorization is calculated for weed images. The colour index was examined with a genetic algorithm and Bayesian categorization (Jin et al., 2021). The deep residual network and the deep dense network are combined in the hybrid deep learning model. The hybrid deep learning model reduces training parameters while increasing accuracy by up to 95.00 % (Zhou et al., 2021).

Deep transfer learning is an amazing performance methodology for identifying plant diseases. For pre-trained datasets, Inception and ImageNet modules were utilized (Chen et al., 2020). The performance of pepper, vegetable, potato, and tomato leaf images in the plantvillage database was studied and enhanced using support vector machine (SVM) and multi-layer perceptron. After training the model the system achieves a higher performance accuracy of 94.35 % (Kurmi et al., 2020).

To detect and recognize corn dietary sickness, a Deep Convolutional Neural Network was deployed., The recognition of corn leaf diseased accuracy was 88.46 %, and the usage of hardware, such as a raspberry pi3 with an Intel Movidius Neural Compute Stick and a system GPU that pre-trained the CNN Model, resulted in superior metric accuracy performance (Sun et al., 2020).

With the rapid growth of artificial intelligence and deep learning technology, computer vision (CV) made a breakthrough. The CV-based approaches are commonly utilized for diagnosing grape leaf diseases. The principle component analysis (PCA) and back propagation methods aid in the diagnosis of grape diseases such as downy mildew and powdery mildew, with a research accuracy of 94.29 % (Xie et al., 2020), using VGGNet. The weights are initialized using ImageNet pre-trained datasets, and over through the real - world dataset, such approaches had a validation accuracy of 91.83 %.

2. Material & methods

2.1. Datasets

To support our research in the area of collection of images available from Pennsylvania state university named plantvillage dataset. The dataset plant-village included 152 crop solutions, 38 crop classes, and 19 crop categories, for 54,303 crop leaves images. In the datasets, high quality JPEG image format with 5471width and 3648 height pixels are available. In the pre-processing, de-nosing, segmentation and after images are 256 X 256 pixels (Gandhi et al., 2018). The plantvillage is a well-known dataset for crop disease, with a large number of public datasets available. A plantvillage dataset images were captured in the lab, thus they are used as training datasets. Our model tested on real field captured images, As a result, we must concentrate on developing our own field database. The test images were captured with a separate Megapixel camera and stored in a database. The datasets prepared in the field are available and be used in the proposed research. The agro-deep mobile application was used to capture some on-field crops images.

The field photographs were taken with the redmi Note 5 Pro MIUI Global 11.0.5.0(PRIMEXM), Android Version PKQ1.180904.001, and a camera frame 4:3 high picture quality on 16 MP+5MP with f/2.2 aperture pixel, in a variety of natural environments. The disease-affected and healthy photos are the most common image categories collected for research purposes. Healthy spot contaminated, mosaic virus, yellow leaf curl virus, septoria leaf spot bacterial spot, early blight, late blight, leaf mould, septoria leaf spot, and spider mites are examples of tomato imagery.

2.2. Proposed research

The schematic in Fig 1 depicts a potential view for multi-crop leaf disease classification and analysis. Initially, plant leaf disease images are collected and classified into several categories. Picture filtering, grey transformation, picture sharpening, and scaling are some of the image-processing techniques. By using data augmentation methods, new sample photos are created from available photos to enhance and prepare the dataset. Augmentation procedures like turning, translation, and randomized transformation are employed to enhance the size of the dataset. The photos are then used as input to the suggested approach for training the model in the following stage. The newly trained architectural model is used to anticipate previously unseen images. Eventually, the findings of plant disease detection and identification are achieved. Finally, complete details of these steps are depicted in later parts (Table 1).

2.3. Sample images category

The sample images of crops shown in Fig. 2 depict the category of field's tomato leaf images of various disease and healthy classes. The images are one-of-a-kind for each type of disease symptom, pattern, spot, and colour mark. Specific tomato plant leaf diseases such as bacterial wilt, leaf mold, and grey spots are identified and detected as disease impacted recognition traits (Paymode et al., 2020).

Fig. 3 depicts field images of grape vine leaves obtained in the Nashik district of Maharashtra, India. A grape category, Healthy 423, Black Rot 1180, Black Measles 1383 and Leaf Blight 1076 images were recorded, recognized, and captured. The datasets for grape plant leaves were generated by adjusting the brightness and hue of images from the A to D category (See Figs. 4-5).

The second crop of tomatoes sampled Early blight 1000, Mosaic virus 373, Bacterial spot 2127, Late blight 1909, Leaf mould 952, Septoria leaf spot 1771, 1404 spot, spider mites 1676 and Yellow leaf curl 3209. The deep learning based methods are state-of-the-art in computer vision, which is used in image recognition and classification. In general, dataset collecting, data pre-processing, image segmentation, feature extraction, and classification are the four stages of Artificial Intelligence (AI) in agriculture

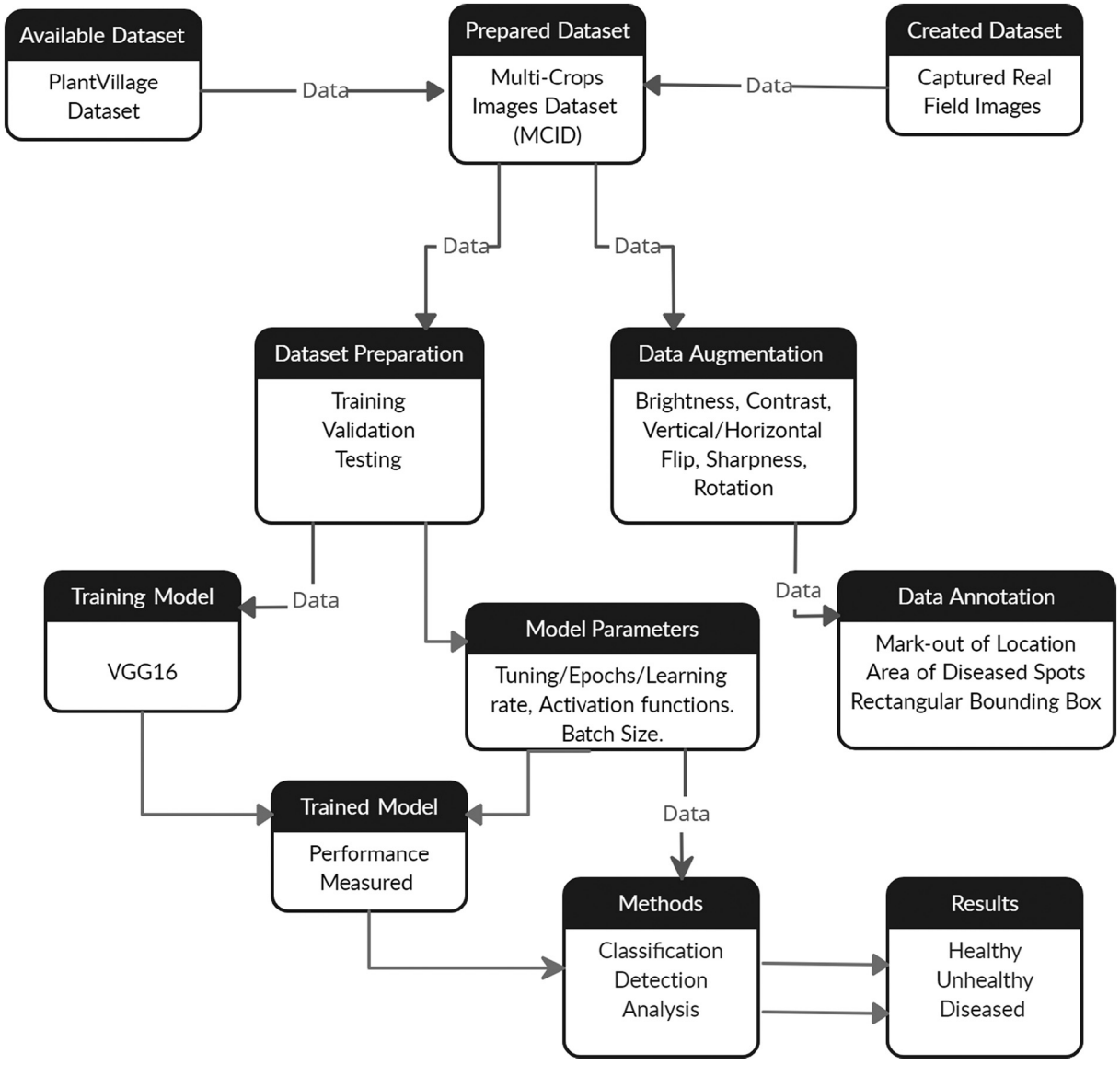


Fig 1. Proposed research system flow diagram.

approaches for crop leaf disease detection and classification utilising Convolutional Neural Network (CNN). A Google Colaboratory platform was used to pre-process the image, extraction of features, and classify it.

2.4. Image augmentation

The large number of datasets improves the learning algorithms' performance and prevents overfitting. Obtaining a real-time dataset for use as input to a training model is a complex and time-consuming operation. As a result, data augmentation broadens the range of training data available to deep learning models. Deep learning-based augmentation approaches include image flipping, cropping, rotation, colour transformation, PCA colour augmentation, noise rejection, Generative Adversarial Networks (GANs), and Neural Style Transfer (NST) (Arun Pandian et al., 2019). The Faster DR-IACNN approach for detecting grape leaf diseases is based on deep learning. The automatic extraction of spots on leaves has a high detection speed and accuracy. There are 4449 original photographs and 62,286 photos developed using data augmentation techniques.

Table 1
A study of deep learning techniques with classification and recognition rate (See Fig. 12).

Approach	Classification	Model	Recognition rate (%)
Hybrid Combination (Massi et al., 2020)	Three SVM	SVM	91.11
Deep Learning (Ferentinos, 2018)	CNN	VGG	97.53
Multi-Context Fusion Network (MCFN) (Wu, 2020)	CNN	AlexNet &VGG16	97.50
Deep Transfer Learning (DTL) (Chen et al., 2020)	CNN	VGG	91.83
Machine Learning (Kurmi et al., 2020)	SVM	MLP	94.35
Deep Learning (Sun et al., 2020)	DCNN	DCNN	88.46
Deep Learning (Xie et al., 2020)	Faster DR-IACNN	Inception-v1 ResNet-v2	81.11

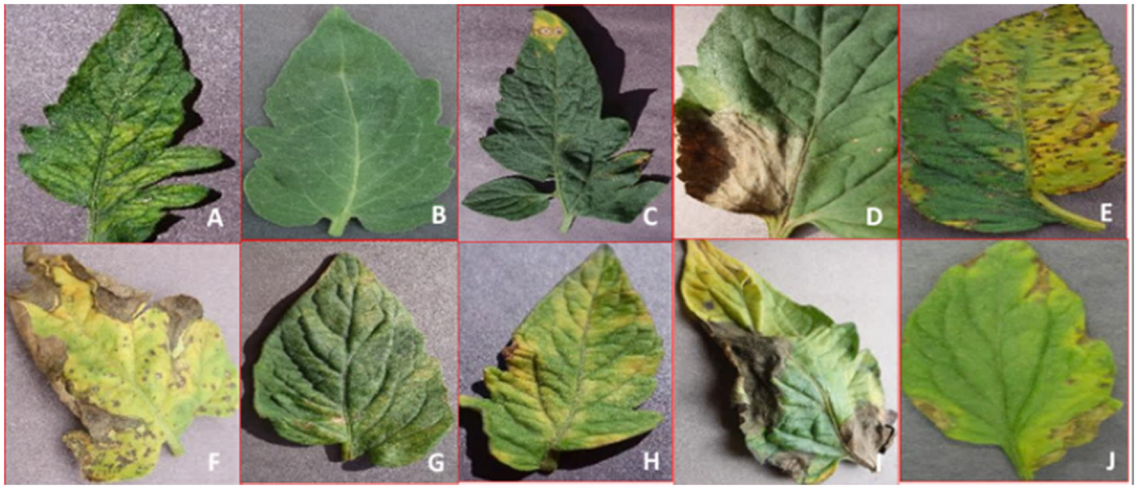


Fig 2. Sample tomato leaf images (A: Mosaic Virus, B: Healthy, C: Target Spot, D: Late Blight, E: Bacterial Spot, F: Septoria Spot, G: Spider Mite, H: Leaf Mold, I: Early Blight, J: Yellow Leaf.

The images are converted into a vector of fixed features through feature extraction in segmentation. The color, texture, and shape are the system-adopted features. A means, confidence intervals, and sleekness have been employed as colored methods, with HSV and RGB color spaces being retrieved. The gray-level co-occurrence matrix is preferred when extracting texture features from a colour image. This approach is used to identify plant diseases.

2.5. Transfer learning

The model's optimization and training is a tough and time-consuming operation. A powerful graphical processing unit (GPU) is required for the training, as well as millions of training examples. However, transfer learning, which is employed in deep learning, solves all of the problems. The pre-trained Convolutional Neural Network (CNN) used in transfer learning is optimized for one task and transfers knowledge to different modes (Nevavuori et al., 2019). The multi-crop image dataset model comprises a size of 224 X 224. The residual network (ResNet) needed to be tweaked. In all ResNet models, the final layer before the softmax is a 7 X 7 average-pooling layer. A smaller image can fit into the network when the pooling size decreases. The basic picture preparation is necessary for the transfer learning considerations with the multi-cropped image dataset.

3. Results & discussion

3.1. Convolutional neural network

The convolutional layers, pooling layers, fully-connected layers, and dense layers constitute the architecture of the Convolutional Neural Network (CNN) (See Fig. 6). The layers' description is shown below.

3.1.1. Convolutional layer

Convolutional layers' fundamental function is to extract unique features from images. The implementation of convolutional layers on a normal basis facilitates the extraction of input features (Chen et al., 2020), The features extraction (Hi) among several layers in CNN is computed using the formula below.

$$H_i = \varphi\left(H_{i-1}\, W_i + b_i\right) \quad (1)$$

Where, Hi - Feature map, Wi –Weight, bi is offset and φ – Rectified Linear Unit (RELU)

3.1.2. Pooling layers

The pooling layers are a crucial component of a Convolutional Neural Network (CNN). It shrinks the size of convolved features in dimension while simultaneously minimizing the computational resources necessary for image processing. Pooling arise categorized into two types: max pooling and average pooling. Max pooling returns the maximum value of images, whereas an average pooling returns the average value of the image section.

3.1.3. Drop-out layers

The dropout layers improve the capability of a trained model. It provides regularization and prevents the model from over-fitting by decreasing the correlation between the neurons. The drop out process is used in all the activation functions but it is scaled by factor (Liu, 2020).

3.1.4. Flatten layers

It collapses the spatial dimensions of the mapped pooled features while retaining the channel dimensions. The flattened layer adds extra dimensions and after it is transformed into a vector. The vectored feed

Fig 3. Sample grapes plant leaf images. (A: Grape Black Rot, B: Grape Esca (Black Measles), C: Healthy, D Grape Leaf blight (Isariopsis Leaf Spot).

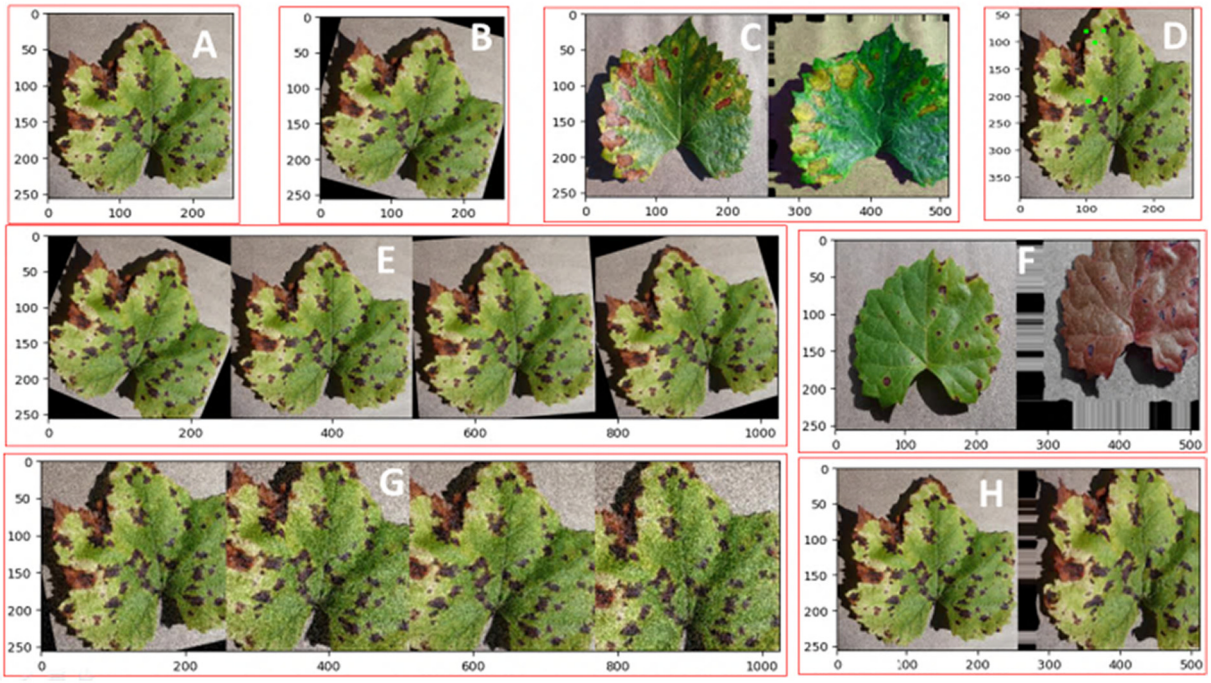


Fig 4. Multi-crops image augmentation (a) (A: Original B: Rotate, C: Color, D: Image Point, E: Hstack, F: Size G: Gaussian Noise, H: Shape).

to fully connected layers also known as the dense layer or fully connected layers.

3.1.5. Fully-connected layers

Fully connected layers are needed for extracted images classification features because of their special purpose. The softmax function predicts earlier extracted image attributes from preceding layers. Softmax is a multiclass classification activation function in the output layers. The neural network layer uses a multilayer perceptron model (MLP) as a classifier for two-class classification. The model with nonlinearity, which is introduced in the full vectors using rectified linear unit (RELU) activation. The versatility of class separation is greater when employing a support vector machine (SVM). The essentials of SVM are as described in the following:

$$Minimize \; \tfrac{1}{2} \sum_{j=1}^{n} W_1^2 + C \sum_{j=1}^{N} \xi j \tag{2}$$

Where C is the tuning measure, subject to the constraint $y_j (\overline{W} \cdot \overline{X} + b) \geq 1 - \zeta, j = 1, 2, 3 \ldots N$. The softmax parameter $\gamma = 1$ and $C = 1$ are used throughout training and test sets of the classification algorithm.

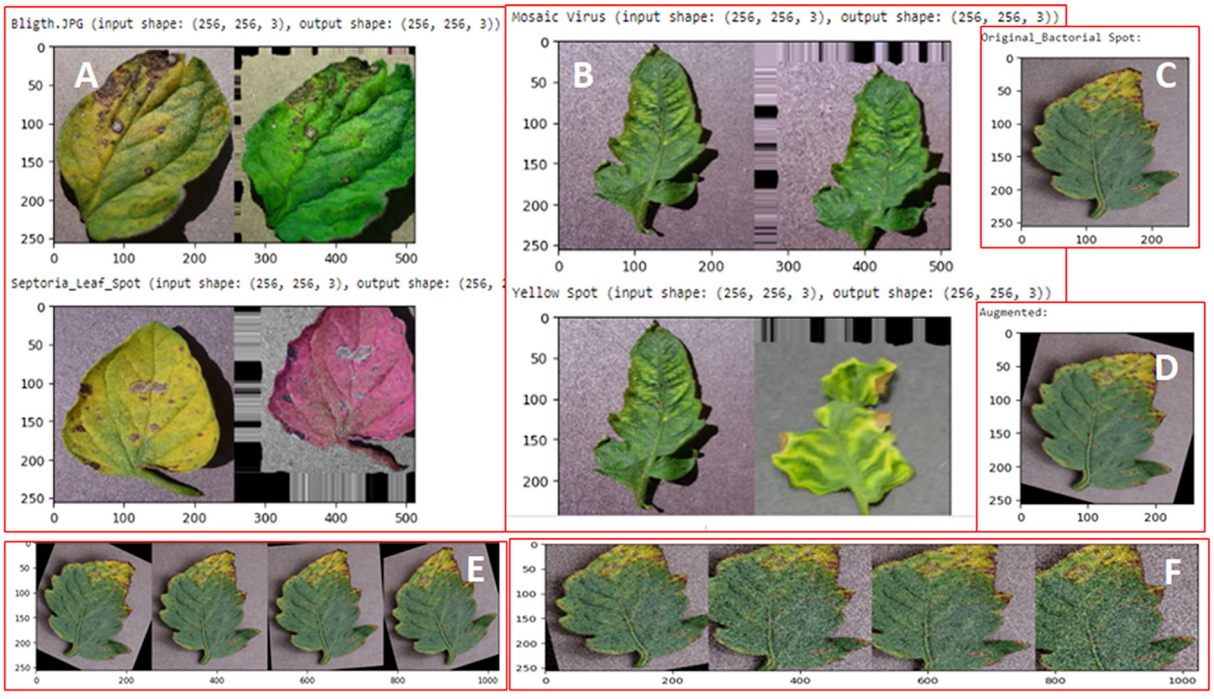


Fig 5. A B: H Stack, C: Original, D: Augmentation, E: Batch H stack, F: Adaptive Gaussians Noise.

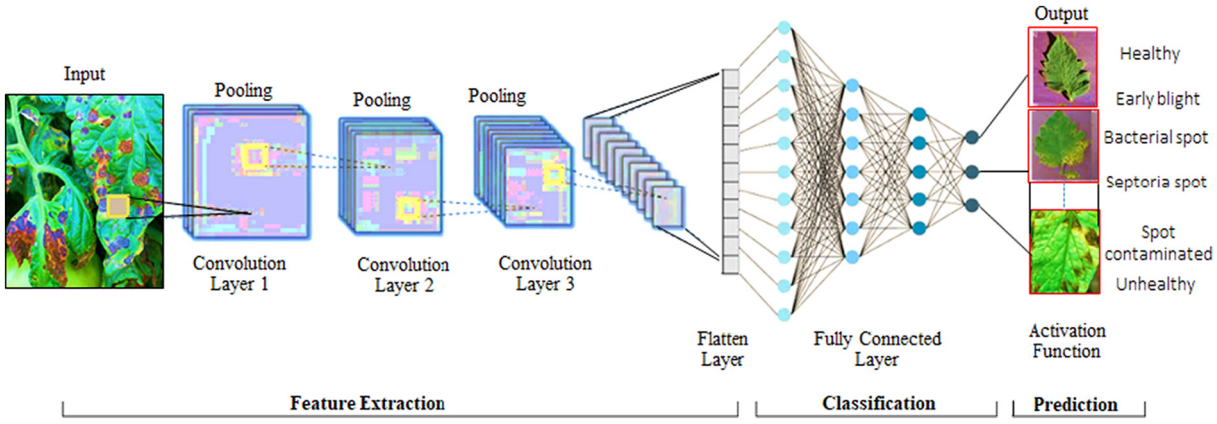


Fig 6. Proposed convolutional neural network CNN architecture.

The ConvNet architecture design's main component is its depth. By defining additional design parameters and growing the network depth continuously, by adding more convolutional layers that are doable by using extremely small (3 x 3) convolution filters in all layers. As a result, they've developed substantially more accurate ConvNet architectures that not only reach state-of-the-art accuracy on fixed dataset classification and localisation tasks, but are also applicable to other image recognition datasets, where they perform admirably even when utilised as part of relatively simple pipelines(Simonyan and Zisserman, 2015).Our ConvNets are fed a fixed-size 224 x 224 RGB picture during training. The only pre-processing we perform is removing each pixel from the mean RGB value determined on the training set. We apply filters with a very small receptive field 3 x 3 to send the image through a stack of convolutional layers. We also use 1 x 1 convolution filters in one of the configurations, which are a linear change of the input channels (followed by non-linearity). The convolution stride is set to 1 pixel, and the spatial padding of the convolutional layer input is set to 1 pixel for 3 conv. layers so that the spatial resolution is kept after convolutional. Five max-pooling layers, which follow part of the convolutional layers, do spatial pooling (not all the convolutional layers are followed by max pooling) Max-pooling is done with stride 2 over a 2 x 2 pixel window.

3.2. VGG16

The Convolutional Neural Network based VGG16 pre-trained models are used to improve the performance and classify the crop images as healthy and disease images. For quality detection and analysis of crop leaf images, the initial model transfers information from pre-trained VGG16 models. The Convolutional Neural Network (CNN) model retained new images of the field and learned to perform a model for disease detection and classification (Alencastre-Miranda et al., 2021).

The VGG model improved with large kernel-sized filters, with 11 and 5 convolutional layers with a 3 x 3-kernel filter size. The input image size is fixed at 224 x 224. Following image pre-processing, images were passed through a convolutional layer with a filter size of (3 x 3).For linear transformation of the input channel, the filter size is set to (1 x 1). The stride size is fixed to 1 and max pooling is performed with 2 x 2 sizes and stride set to 2. In the next steps, fully connected layers have the same configuration with 4096 channels in each layer. The final layer is the softmax activation layers, followed by the RELU activation functions (See Fig. 7).

3.3. Performance measure

The F1 score, accuracy matrix, and Receiver operating characteristic (ROC), as well as the area under the curve (AUC), are being used to evaluate segmentation performance (AUC). The performance of the classifier is measured using evaluation metrics.

3.3.1. Accuracy metrics

The model performance for all classes is accurately measured. The accuracy is calculated by adding the total number of correct predictions to the total number of predictions. The performance parameter calculation of precision and recall and F1-Score are measured. The accuracy is expressed in terms as follows.

$$AC = \frac{(TP + TN)}{(TP + FP + FN + TN)} \tag{3}$$

Where, TP is True Positive, TN True Negative, FN False Negative and FP False Positive Samples. The classifier performance measure using evaluation metrics are gives as;

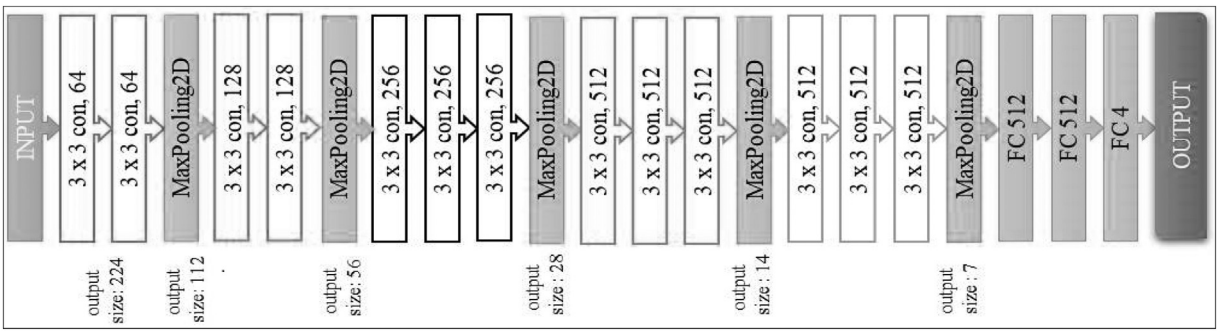


Fig 7. Proposed convolutional neural network (CNN) VGG16 architecture.

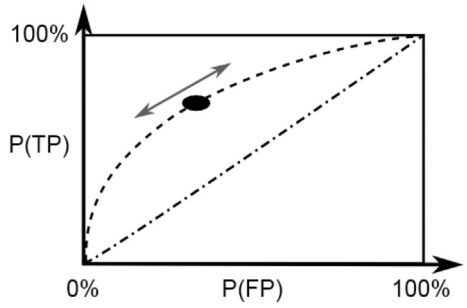


Fig 8. Receiver operating characteristics FP versus TP

Table 2
Parameter setting for trained the model

Hyperparameter	Value Setting
Crops	Grapes & Tomatoes
Convolutional Layers	13
Max Pooling Layer	5
Dropout Rate	015/0.25/0.50
Activation Function	Relu, Softmax
Epochs	20/25/30/40/45
Learning Rate	0.00001/0.0001
Image Size	224 x 224 x 3

$$TPR = (Sensitivity) = \frac{TP}{(TP + FN)}, TNR = (Specificity)\ \frac{TN}{(TN + FP)} \tag{4}$$

$$FPR = \frac{FP}{(FP + TN)} \tag{5}$$

Where, TPR is True Positive Rate, TNR True Negative Rate, and FPR False Positive Rate.

$$Precision = \frac{TP}{(TP + FP)}, Recall = \frac{TP}{(TN + FN)} \tag{6}$$

$$G{-}Mean = \left(\prod_{K=1}^{m} Recall_K \right) \frac{1}{m} \tag{7}$$

Here m represents the number of categories and G denotes the TNR and FPR accuracy ratio.

Mean average precision (mAP), which consists of Precision, Recall, and Mean, is the algorithm assessment standard employed. Image processing and detection rely heavily on the mAP. From the entire results, the accuracy has classified correctly. From the complete findings, the recall is correctly classified.

The F1 score is another important metric for evaluating the algorithm. It's precision and recall fundamental that's presented as follows:

$$F1\ Score = \frac{2 \times Precision \times Recall}{(Precision + Recall)} \tag{8}$$

3.3.2. Receiver operating characteristic

The receiver operating characteristic (ROC) curve is used to understand deterministic indications of categorization sorting as well as computational modeling challenges. The curve is a graph that shows the ratio of false positives to true positives under different standard limits (See Fig. 8).

A prototype also with largest true negative rate values was used to correctly categorize defectives, and the model with the highest true positive rate values was used to correctly classify healthily. To boost productivity by reducing processing time for training and testing, the MCC (Matthews Correlation Coefficient) is employed for the total computation. MCC is a criterion for categorizing complex data into distinct categories. MCC is a superior method to accuracy which only has significant importance if the true positives, true negatives, false negatives, and false positives outcomes are all positive. The MCC ranges from 1 (poorest judgment) to 1 (perfect predictions), with an MCC of 0 suggesting a random guess.

The model is tuned by the number of epochs, hidden layers, hidden nodes, activation functions, dropout, learning rates, and batch size. The model performance is affected by hyper parameter tuning. The term "hyper parameter tuning" refers to the process of repeatedly adjusting hidden layers, epochs, activation function, or learning rate. The model is fine-tuned to achieve the best accuracy while minimising the average loss.

The experimental analysis was carried out on Google research products on Google Colaboratory. The Colaboratory platform supports python programming, and nearly all of the Python libraries are uploaded and installed for research purposes. The Python 3 Google Compute Engine backend (GPU) with RAM of 12.72 GB and disc space of 68.40 GB is available while experimenting. The dataset is uploaded with the drive mounted, and the model is trained on the Google platform with high configurations. A Python convert to image function is used for converting all the images to an array and fetching images from the directory.

The processed images come from a directory, and all label images are transformed using the label binarized sklearn python package. The total number of images is divided into train and test using train-test-split python functions. The model parameters were set as shown in Table 2, and the model was trained to calculate all trainable and non-trainable parameters. The Adam optimization algorithm is used to train the deep learning convolutional neural network model. The algorithm optimized the sparse gradient noise issue.

The input network uses 224 X 224 images, and the batch size is 30 for grapes and 25 for tomatoes, respectively, and the same test is performed for different epochs with batch size and learning rate. In every polling layer with a 2 x 2-pool size and the RELU function utilized in the network, the model performs a max-pooling operation. The output of the last layer is a oftmax-activation multi-crop-developed prediction. During the network's training phase, hyper parameters such as learning rate and epoch size were adjusted. The average accuracy achieved was 98.40% for grapes and 95.71% for tomatoes, respectively. The learning rate is tested at different values to optimize targeted performance measured. The validation process is based on a total number of images from the multi-crop dataset. With the setting of different epochs and batch size, the accuracy improved and grew.

The crops-leaf images datasets are used to train the model and identification of class and category of disease with transfer learning techniques including VGG16. The original datasets are divided into training data 80%, validation data 10% and testing data 10%.

Table 3
Experimental results of the grape model for setting different parameters.

No. of epochs	Learning rate	Dropout rate	No. of images	Training loss	Training accuracy	Validation loss	Validation accuracy
40	0.00001	0.25	450	0.0897	0.9840	0.0486	0.9889
30	0.0001	0.50	450	0.1136	0.9585	0.0686	0.9867
45	0.00001	0.25	400	0.0995	0.9796	0.0529	0.9858
45	0.0001	0.25	750	0.0875	0.9696	0.0521	0.9853
30	0.0001	0.50	450	0.1326	0.957	0.0686	0.9843
40	0.001	0.30	600	0.1139	0.9606	0.0529	0.9831

Table 4
Experimental results of the tomatoes model for setting different parameters.

No. of epochs	Learning rate	Dropout rate	No. of images	Training loss	Training accuracy	Validation loss	Validation accuracy
25	0.0001	0.25	200	0.1643	0.9571	0.2627	0.9432
35	0.00001	0.20	180	0.2203	0.9281	0.3143	0.9013
30	0.00001	0.15	200	0.2624	0.9097	0.3345	0.8926
30	0.00001	0.25	200	0.3042	0.8983	0.3736	0.8849
30	0.00001	0.25	180	0.4871	0.8354	0.4149	0.8671
30	0.00001	0.50	200	0.5226	0.8255	0.4508	0.8538

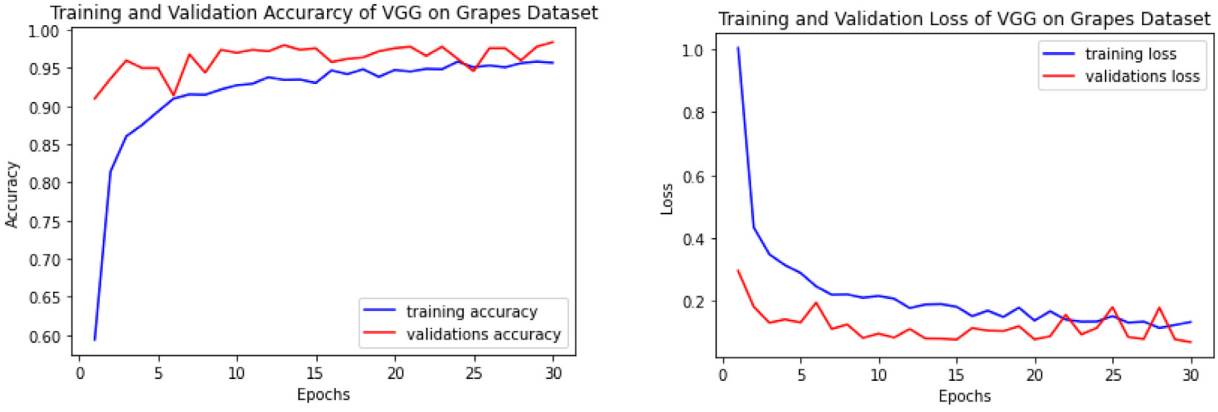
3.4. Training and validation accuracy

Training and validation accuracy is measured by setting different values while training the model. The experiments were carried out at Google Colaboratory on the available RAM of 12.50 GB. While performing the experiment, different values are set for the following: the number of epochs, learning rate, dropout rate, and the number of images noted as training loss, training accuracy, validation loss, and validation accuracy. A model's performance is measured and verified on the grape and tomato crops' leaves. Table 3 and Table 4 show the details of the results of experiments carried on grapes and tomatoes, respectively.

3.5. Figures and graphs

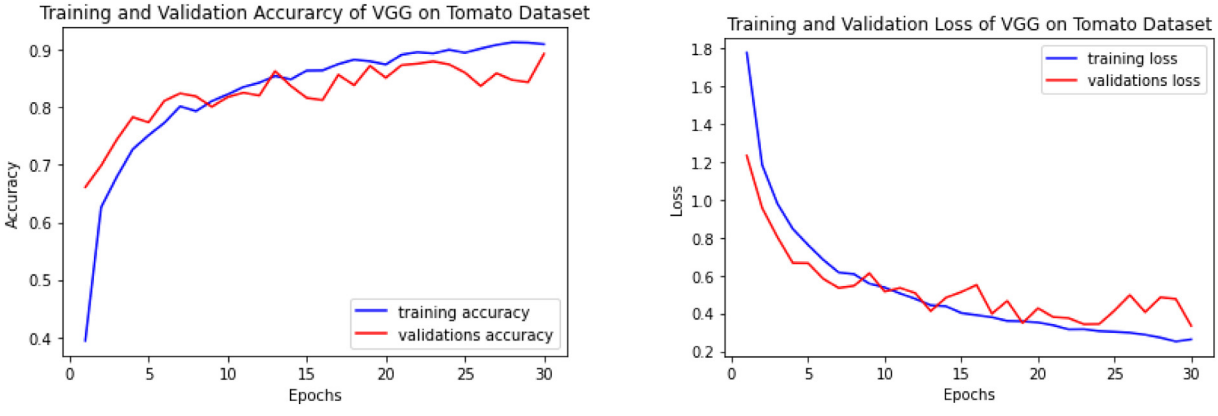
A model's performance is measured and verified with training, testing, and validation methods for grapes and tomatoes leaves. Fig 9 and Fig. 10 show the training and validation accuracy and loss of the grape leaves and tomatoes, respectively.

The confusion matrix has been used to measure the performance parameter for grapes and tomatoes leaves, as shown in Fig. 11. Experiment with the facts collected. The suggested approach is tested using our grapes and tomatoes image datasets, which were taken in a real-field with various backdrop and light intensities, similar to the tests done in Section 4.4.



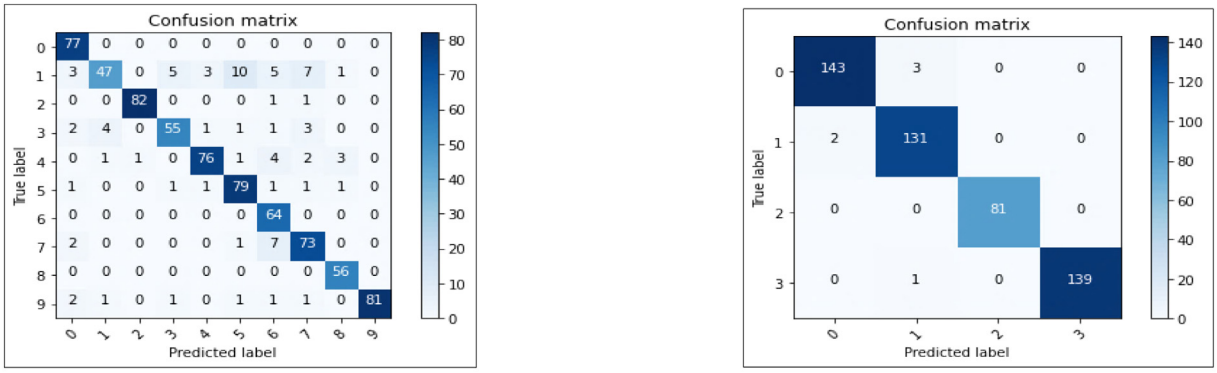
(a) (b)

Fig 9. Training and validation. (a) Accuracy and (b) loss of VGG16 grapes.



(a) (b)

Fig 10. Training and validation. (a) Accuracy and (b) Loss of VGG16 tomatoes.



(a) (b)

Fig 11. Confusion matrix. (a) Tomatoes and (b) grapes.

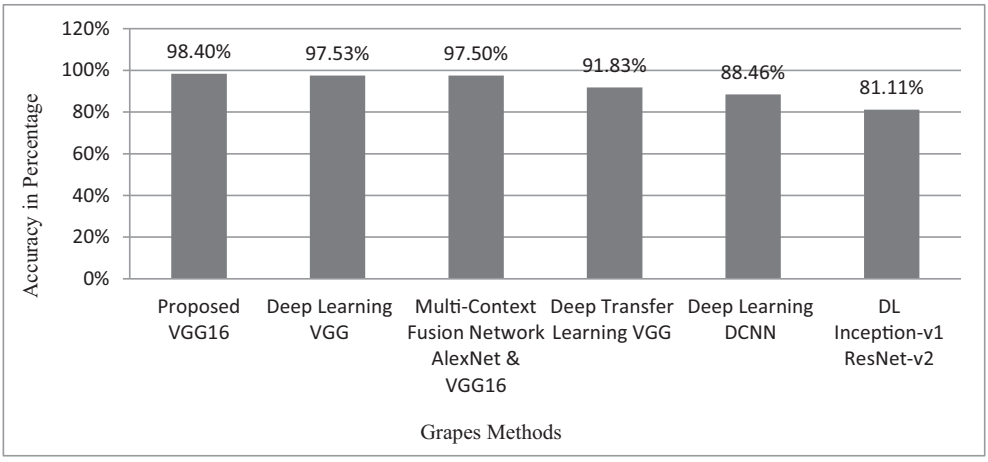


Fig 12. Comparison between different model vs accuracy in percentage(%) with proposed VGG16 grapes.

To assure the diversity of sample images and avoid the over fitting problem, data augmentation techniques such as random rotation, flipping, and scale transform, as well as associated pre-processing activities, are used to extend the training samples. The processes are described in more detail below.

1. Image resize: The total images scaled into size of 224 x 224 pixels, for the model fit and minimum 200 images taken from each healthy and unhealthy category are augmented with data augmentation methods.
2. Image pre-processing: Image pre-processing is used to darken the different lengths of the image data, going to bring them into ratio

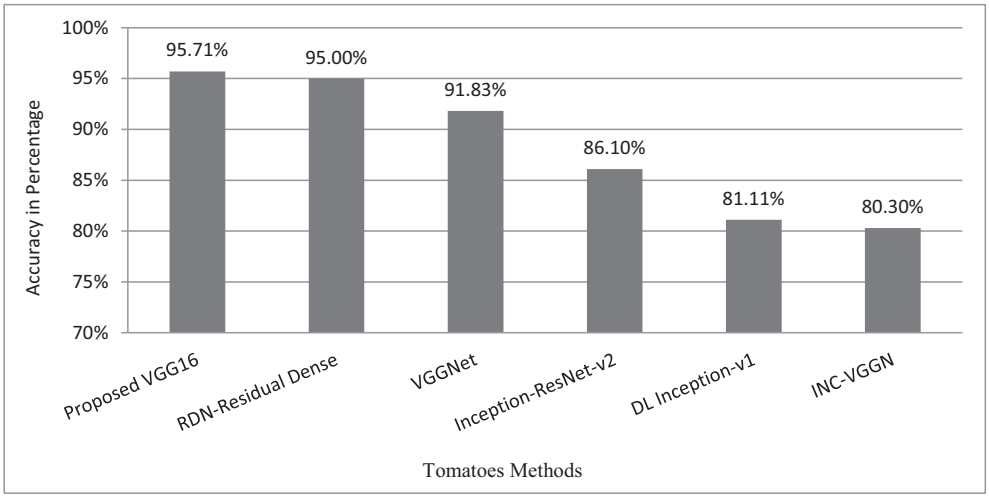


Fig 13. Comparison between different model vs accuracy in percentage (%) with proposed VGG16 tomatoes.

and retaining the initial images' knowledge formation while attempting to prevent image deformation.
3. The dataset partition and training. In this section a selection of random sample images for proposed experiments and calculated with carried out the result as per Section 4.4.
4. Validation and testing. The testing is done on the images that were used to evaluate the model, and new images from outside modeling are used to check the model effectiveness. The output results are compared to the real categories, the effectiveness of the control that goes with them is computed.

The residual block collection and DesnseNet used in task of tomato leaf disease identification with RDN restructured model. After input image normalizing and adding the convolutional layer residual modules dense layer classify the tomato disease images with 95% accuracy disease dataset (Zhou et al., 2021). The public data set of the AI Challenger Competition in 2018 used the Inception-ResNet-v2 model using the RELU activation function, with an accuracy of 86.1%. (Ai et al., 2020), under complex background conditions, the accuracy of VGG Net is 91.83 %. One more approach to INC-VGGN rice disease detection with an average accuracy of 80.38% for both "Phaeo- sphaeria Spot" and "Maize Eyespot" diseases (J. Chen et al., 2020) (See Fig. 13).

4. Conclusion

In this paper, there are two types of crop disease leaves were collected and prepared as a dataset with available data. The techniques of data augmentation, dataset pre-processing, training, and testing are applied to the convolutional neural network-based VGG16 model. The proposed model is built and tested to improve the performance measured and compared. The evaluation metrics parameters are higher and increased as compared to other available datasets and methods. Therefore, our proposed research work increased accuracy for grapes by 98.40% and for tomatoes by 95.71%. Always improving the performance of on-field crops, leaf images and diseases classification and analysis is a critical step, but with our model achieved the highest performance, which supported agricultural development. The major focus of research is to provide advancement in the agriculture sector and an increase in food production. The collection and preparation of genuine datasets and applying to the deep learning models with multiple crops leaves images is a future target. In the future, the use of Inception V3 and ResNet-based CNN models for much deeper analysis of crop images is anticipated. Our work encourages and stimulates farmers, which ultimately raises farm income and helps to build up powerful countries.

Acknowledgements

Farmers from Nashik and Aurangabad, Maharashtra [India], contributed to the collection of real field crop images for research purposes. We would like to thank you Dr. Panjabrao Deshmukh Krushi Vidyapeet (Dr. PDKV), Akola, Maharashtra [India], for their encouragement and assistance.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Credit author statement

We the undersigned declare that this manuscript is original, has not been published before and is not currently being considered for publication elsewhere.

We would like to draw the attention of the Editor to the following publications of one or more of us that refer to aspects of the manuscript presently being submitted. Where relevant copies of such publications are attached

We wish to draw the attention of the Editor to the following facts which may be considered as potential conflicts of interest and to significant financial contributions to this work. [OR]

We wish to confirm that there are no known conflicts of interest associated with this publication and there has been no significant financial support for this work that could have influenced its outcome.

We confirm that the manuscript has been read and approved by all named authors and that there are no other persons who satisfied the criteria for authorship but are not listed. We further confirm that the order of authors listed in the manuscript has been approved by all of us.

We confirm that we have given due consideration to the protection of intellectual property associated with this work and that there are no impediments to publication, including the timing of publication, with respect to intellectual property. In so doing we confirm that we have followed the regulations of our institutions concerning intellectual property.

We further confirm that any aspect of the work covered in this manuscript that has involved either experimental human patients has been conducted with the ethical approval of all relevant bodies and that such approvals are acknowledged within the manuscript.

We understand that the Corresponding Author is the sole contact for the Editorial process (including Editorial Manager and direct communications with the office). He/she is responsible for communicating with the other authors about progress, submissions of revisions and final approval of proofs. We confirm that we have provided a current, correct email address which is accessible by the Corresponding Author and which has been configured to accept email from biomaterials@elsevier.com.

References

Ai, Y., Sun, C., Tie, J., Cai, X., 2020. Research on recognition model of crop diseases and insect pests based on deep learning in harsh environments. IEEE Access 8, 171686–171693. https://doi.org/10.1109/access.2020.3025325.

Alencastre-Miranda, M., Johnson, R.M., Krebs, H.I., 2021. Convolutional neural networks and transfer learning for quality inspection of different sugarcane varieties. IEEE Trans. Indust. Inform. 17 (2), 787–794. https://doi.org/10.1109/TII.2020.2992229.

Ang, K.L.M., Seng, J.K.P., 2021. Big data and machine learning with hyperspectral information in agriculture. IEEE Access 9, 36699–36718. https://doi.org/10.1109/ACCESS.2021.3051196.

Arun Pandian, J., Geetharamani, G., Annette, B., 2019. Data augmentation on plant leaf disease image dataset using image manipulation and deep learning techniques. Proceedings of the 2019 IEEE 9th International Conference on Advanced Computing, IACC 2019, pp. 199–204 https://doi.org/10.1109/IACC48062.2019.8971580.

Chen, J., Chen, J., Zhang, D., Sun, Y., Nanehkaran, Y.A., 2020. Using deep transfer learning for image-based plant disease identification. Comput. Electr. Agricult. 173 (November 2019) 105393. https://doi.org/10.1016/j.compag.2020.105393.

Chen, C.J., Huang, Y.Y., Li, Y.S., Chen, Y.C., Chang, C.Y., Huang, Y.M., 2021. Identification of fruit tree pests with deep learning on embedded drone to achieve accurate pesticide spraying. IEEE Access 9, 21986–21997. https://doi.org/10.1109/ACCESS.2021.3056082.

Ferentinos, K.P., 2018. Deep learning models for plant disease detection and diagnosis. Comput. Elect. Agric. 145 (September 2017), 311–318. https://doi.org/10.1016/j.compag.2018.01.009.

Gandhi, R., Nimbalkar, S., Yelamanchili, N., Ponkshe, S., 2018. Plant disease detection using CNNs and GANs as an augmentative approach. 2018 IEEE International Conference on Innovative Research and Development, ICIRD 2018, no. May: 1–5 https://doi.org/10.1109/ICIRD.2018.8376321.

Hiary, H., Al, S.B., Ahmad, M., Reyalat, M. Braik, ALRahamneh, Z., 2011. Fast and accurate detection and classification of plant diseases. Int. J. Comput. Appl. 17 (1), 31–38. https://doi.org/10.5120/2183-2754.

Jin, X., Che, J., Chen, Y., 2021. Weed identification using deep learning and image processing in vegetable plantation. IEEE Access 9, 10940–10950. https://doi.org/10.1109/ACCESS.2021.3050296.

Kurmi, Y., Gangwar, S., Agrawal, D., Kumar, S., Srivastava, H.S., 2020. Leaf image analysis-based crop diseases classification. Signal Image Video Process. https://doi.org/10.1007/s11760-020-01780-7.

Liu, S.Y., 2020. Artificial intelligence (AI) in agriculture. IT Profes. 22 (3), 14–15. https://doi.org/10.1109/MITP.2020.2986121.

Massi, I.E., Mostafa, Y.E.-s., Yassa, E., Mammass, D., 2020. Combination of multiple classifiers for automatic recognition of diseases and damages on plant leaves. Signal Image Video Process. https://doi.org/10.1007/s11760-020-01797-y.

Nevavuori, P., Narra, N., Lipping, T., 2019. Crop yield prediction with deep convolutional neural networks. Comput. Elect. Agric. 163 (June) 104859. https://doi.org/10.1016/j.compag.2019.104859.

Paymode, A.S., Malode, V.B., Shinde, U.B., 2020. Artificial intelligence in agriculture for leaf disease detection and prediction: a review 13 (4), 3565–3573.

Simonyan, K., Zisserman, A., 2015. Very deep convolutional networks for large-scale image recognition. 3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings . http://www.robots.ox.ac.uk/.

Sun, J., Yang, Y., He, X., Xiaohong, W., 2020. Northern maize leaf blight detection under complex field environment based on deep learning. IEEE Access 8, 33679–33688. https://doi.org/10.1109/ACCESS.2020.2973658.

Wu, W., Yang, T.L., Li, R., Chen, C., Liu, T., Zhou, K., Sun, C.M., Li, C.Y., Zhu, X.K., Guo, W.S., 2020. Detection and enumeration of wheat grains based on a deep learning method under various scenarios and scales. J. Integr. Agric. 19 (8). https://doi.org/10.1016/S2095-3119(19)62803-0.

Xie, X., Ma, Y., Liu, B., He, J., Li, S., Wang, H., 2020. A deep-learning-based real-time detector for grape leaf diseases using improved convolutional neural networks. Front. Plant Sci. 11 (June) 1–14. https://doi.org/10.3389/fpls.2020.00751.

Zhou, C., Zhou, S., Xing, J., Song, J., 2021. Tomato leaf disease identification by restructured deep residual dense network. IEEE Access 9, 28822–28831. https://doi.org/10.1109/ACCESS.2021.3058947.