DEEP LEARNING BASED PLANT DISEASE DETECTION FOR AGRICULTURAL HEALTH MONITORING

Mini Project submitted in partial fulfillment of the requirement for the award of the degree

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

Under the esteemed guidance of

Mrs. G. Santhoshi

Assistant Professor

By

Berelli Preetham Rao	**21R11A05B1**
Mamilla Rithika	**21R11A05D0**
Manda Anjani	**21R11A05D1**

Department of Computer Science and Engineering

Accredited by NBA

Geethanjali College of Engineering and Technology

(UGC Autonomous)

(Affiliated to J.N.T.U.H, Approved by AICTE, New Delhi)

Cheeryal (V), Keesara (M), Medchal.Dist.-501 301.

September-2024

Geethanjali College of Engineering & Technology

(UGC Autonomous)

(Affiliated to JNTUH, Approved by AICTE, New Delhi)
Cheeryal (V), Keesara(M), Medchal Dist.-501 301.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
Accredited by NBA

CERTIFICATE

This is to certify that the B.Tech Mini Project report entitled **"Deep Learning Based Plant Disease Detection for Agricultural Health Monitoring"** is a bonafide work done by **Berelli Preetham Rao (21R11A05B1), Mamilla Rithika (21R11A05D0), Manda Anjani (21R11A05D1)**, in partial fulfillment of the requirement of the award for the degree of Bachelor of Technology in "**Computer Science and Engineering**" from Jawaharlal Nehru Technological University, Hyderabad during the year 2024-2025.

Internal Guide | HOD – CSE

G. Santhoshi
Assistant Professor

Dr A SreeLakshmi
Professor

External Examiner

Geethanjali College of Engineering & Technology

(UGC Autonomous)

(Affiliated to JNTUH Approved by AICTE, New Delhi)
Cheeryal (V), Keesara(M), Medchal Dist.-501 301.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Accredited by NBA

DECLARATION BY THE CANDIDATE

We, **Berelli Preetham Rao, Mamilla Rithika, Manda Anjani**, bearing Roll Nos. **21R11A05B1, 21R11A05D0**, **21R11A05D1**, hereby declare that the project report entitled **"Deep Learning Based Plant Disease Detection for Agricultural Health Monitoring"** is done under the guidance of **Mrs. G. Santhoshi, Assistant Professor,** Department of Computer Science and Engineering, Geethanjali College of Engineering and Technology, is submitted in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology in Computer Science and Engineering**.

This is a record of bonafide work carried out by us in **Geethanjali College of Engineering and Technology** and the results embodied in this project have not been reproduced or copied from any source. The results embodied in this project report have not been submitted to any other University or Institute for the award of any other degree or diploma.

Berelli Preetham Rao (21R11A05B1)

Mamilla Rithika (21R11A05D0)

Manda Anjani (21R11A05D1)

Department of CSE,

Geethanjali College of Engineering and Technology, Cheeryal.

ACKNOWLEDGEMENT

We, the students of the Computer Science and Engineering Department, are greatly indebted to the authorities of Geethanjali College of Engineering and Technology, for providing us with the necessary facilities to successfully carry out this seminar work in our project titled **"Deep Learning Based Plant Disease Detection for Agricultural Health Monitoring".**

We extend our heartfelt thanks to **Prof, Dr. S. Udaya Kumar,** Principal of Geethanjali College of Engineering and Technology, for his invaluable guidance and continuous encouragement. His insightful advice and dedication to fostering an environment of innovation and learning have empowered us to successfully bring this project to fruition.

We are profoundly grateful to **Dr. A. SreeLakshmi, Professor,** Head of the Department of Computer Science and Engineering, for her steadfast commitment to nurturing our academic and professional growth. Her comprehensive guidance and dedication to our career development in the field of Computer Science have been a source of great inspiration.

Our sincere thanks go to **Mrs. G. Santhoshi, Assistant Professor**, whose expert guidance and technical expertise have been invaluable in steering us through the complex aspects of our project. Her willingness to share her knowledge and provide constructive feedback has significantly contributed to the success of our technical work.

Finally, we would like to express our heartfelt thanks to our parents who were very supportive both financially and mentally and for their encouragement to achieve our set goals.

Berelli Preetham Rao (21R11A05B1)

Mamilla Rithika (21R11A05D0)

Manda Anjani (21R11A05D1)

ABSTRACT

Plant diseases are a major concern for farmers worldwide, as they can lead to significant crop loss and reduced agricultural productivity if not identified and managed in time. Traditional methods of disease detection often rely on manual inspection, which can be slow, labour-intensive, and prone to errors. Early and accurate diagnosis is crucial for preventing the spread of diseases and ensuring optimal crop health, but achieving this in a scalable and efficient manner remains a challenge, especially for large-scale farming operations.

To address this issue, we developed a deep learning-based plant disease detection system using Convolutional Neural Networks (CNNs). Our system classifies leaf images into 39 distinct disease categories based on visible symptoms such as spots, discoloration, and texture changes. Built using the PyTorch framework, the model is trained on a large and diverse dataset of plant images, which allows it to accurately diagnose a wide range of plant diseases. The system automates the detection process, enabling real-time identification and reducing the dependency on manual labour or expert knowledge.

The CNN architecture comprises multiple convolutional layers for feature extraction, pooling layers for down-sampling, and fully connected layers for classification. Data augmentation techniques are applied to enhance the model's robustness to varying image conditions. This approach ensures high accuracy and adaptability across different plant species and disease types. By providing an efficient and reliable solution for early plant disease detection, our system supports farmers in making timely decisions, ultimately helping to reduce crop loss and improve agricultural productivity.

LIST OF FIGURES/DIAGRAMS/GRAPHS

LIST OF ABBREVIATIONS

S. No	Abbreviation	Full Form
1	CNN	Convolutional Neural Networks
2	HTML	Hypertext Markup Language
3	CSS	Cascading Style sheets
4	JS	Java Script
5	API	Application Programming Interface
6	ReLU	Rectified Linear Unit

TABLE OF CONTENTS

1. INTRODUTION

1.1 BACKGROUND

Plant diseases pose a significant threat to global agriculture, affecting both crop yield and quality, with the potential to spread rapidly and cause severe economic losses. Early detection is critical to prevent these issues from escalating, but timely and accurate identification remains a challenge, especially in remote or under-resourced areas with limited access to agricultural experts. Traditional methods rely on visual inspections by farmers or specialists, which are subjective and prone to error, as many plant diseases share similar symptoms like leaf spots, discoloration, and wilting. The complexity and variety of plant diseases further complicate accurate diagnosis based on visual cues alone. Moreover, delays in diagnosis can allow diseases to spread unchecked, while misdiagnosis may lead to inappropriate treatments that exacerbate the problem.

With the growing demand for food production to meet the needs of a rising global population, it is more important than ever to address the challenges associated with plant disease detection. The current methods, which are labor-intensive and time-consuming, are insufficient for large-scale farming operations. There is a critical need for more efficient and reliable approaches that can provide real-time, accurate diagnosis of plant diseases, enabling farmers to take immediate action to protect their crops and ensure sustainable agricultural practices.

Fig 1.1.1 Potato leaf infected with a fungal blight

1.2 OBJECTIVES

1. To develop an efficient and accurate system for detecting plant diseases through image-based analysis, significantly reducing the time needed for diagnosis. This system aims to provide reliable results that can help farmers take timely action against potential threats to their crops.
2. To automate the identification of 39 distinct plant disease categories, allowing the system to recognize various visual symptoms such as leaf spots, discoloration, and wilting. This automation will ensure comprehensive coverage of common plant diseases affecting a wide range of crops.
3. To leverage deep learning techniques, particularly Convolutional Neural Networks (CNNs), to enhance the accuracy of disease classification. CNNs are capable of automatically learning and extracting relevant features from images, leading to improved diagnostic capabilities. This approach will enable the model to effectively discern complex patterns associated with different plant diseases.
4. To provide a scalable solution that can be implemented in real-time, enabling farmers to quickly diagnose plant diseases by capturing and uploading leaf images. This capability ensures that farmers can receive instant feedback on plant health, allowing for prompt interventions.
5. To ensure that the system generalizes well across different plant species and environmental conditions by training on a diverse dataset of leaf images. A robust training process will enhance the model's ability to accurately diagnose diseases in various settings.
6. To reduce reliance on manual detection methods, minimizing human error and the need for expert intervention while offering consistent and objective results.
7. To facilitate early detection and diagnosis of plant diseases, helping to prevent their spread and reduce overall crop loss for improved agricultural productivity.
8. To create a user-friendly interface that allows farmers with minimal technical expertise to easily upload images for instant diagnosis, supporting timely decision-making.

2. SYSTEM ANALYSIS

2.1 EXISTING SYSTEM

Current methods for plant disease detection primarily rely on manual inspection and visual assessment by farmers or agricultural experts, which involves examining the leaves, stems, and overall health of plants for signs of disease such as discoloration, spots, or wilting. While farmers often use their experience to identify potential issues, this approach can lead to inaccuracies, especially when symptoms are subtle or similar across multiple diseases. Some farmers also utilize field guides or mobile applications that provide general information about plant diseases; however, these resources may not cover the full range of diseases or offer detailed identification methods. Additionally, agricultural extensions may provide diagnostic services where plant samples are sent to laboratories for analysis, but this process can be time-consuming and costly, resulting in delays that allow diseases to spread further. These traditional systems are limited by subjectivity and accessibility, making timely responses difficult for farmers. Overall, while existing methods can be effective in certain scenarios, they often lack the speed, accuracy, and scalability needed to effectively address modern agricultural challenges.

Drawbacks of Existing System

- Reliance on manual inspection leads to subjective assessments, resulting in inconsistent diagnoses and potential misidentification of diseases.
- Visual assessments are time-consuming, especially for large fields, causing delays in timely intervention and allowing diseases to spread.
- Field guides and mobile applications may not provide comprehensive coverage of all possible diseases, limiting their effectiveness in accurate identification.
- Diagnostic services that require sending samples to laboratories can be costly and slow, prolonging the wait for results and hindering timely decision-making.
- Limited access to agricultural experts in rural or underserved areas restricts timely assistance and advice for farmers, complicating effective disease management.

2.2 PROPOSED SYSTEM

The proposed system aims to revolutionize plant disease detection by utilizing advanced technologies such as deep learning and image processing to provide timely, accurate, and scalable diagnoses. By employing Convolutional Neural Networks (CNNs), the system will analyze images of plant leaves to automatically identify various plant diseases based on visual symptoms like spots, discoloration, and wilting. Farmers will be able to use a mobile application to capture and upload images of affected leaves, receiving instant feedback on potential diseases through an intuitive user interface. Leveraging a comprehensive dataset of labeled leaf images, the CNN model will be trained to recognize and classify up to 39 distinct plant disease categories, enhancing its diagnostic capabilities. The system will also provide actionable insights and recommendations for disease management, empowering farmers to take prompt and informed actions. By automating the disease detection process, the proposed system aims to reduce reliance on manual inspections and expert intervention, minimizing human error and inconsistencies in diagnoses, ultimately improving crop management, reducing losses due to diseases, and enhancing agricultural productivity.

Advantages of Proposed System

- Provides timely and accurate diagnoses, allowing farmers to quickly identify and address potential threats to their crops.
- Utilizes deep learning algorithms for high-precision image analysis, significantly reducing the chances of misidentification compared to manual inspections.
- Features a user-friendly mobile application, making it accessible for farmers of varying technical expertise to independently diagnose plant diseases.
- Capable of classifying up to 39 different plant disease categories, ensuring comprehensive coverage for recognizing a wide range of diseases.
- Automates the disease detection process, minimizing human error and leading to more reliable outcomes and better decision-making.

2.3 FEASIBILITY STUDY

2.3.1 Details

The proposed system for plant disease detection using deep learning and image processing aims to provide farmers with a practical tool to diagnose plant diseases accurately and efficiently. By employing Convolutional Neural Networks (CNNs), the system will analyze images captured by users via a mobile application. The application will facilitate real-time feedback, enabling farmers to take prompt action against identified diseases. The project will involve developing a comprehensive dataset of labeled leaf images for training the CNN model, ensuring it can recognize a variety of diseases across different plant species.

2.3.2 Impact on Environment

The implementation of the proposed system is expected to have a positive impact on the environment by promoting sustainable agricultural practices. Early detection of plant diseases can lead to reduced reliance on chemical pesticides, as farmers can treat affected plants more precisely and effectively. This targeted approach minimizes chemical runoff and its associated environmental harm. Additionally, by improving crop health and yields, the system supports food security and reduces waste, contributing to overall ecosystem stability.

2.3.3 Safety

The proposed system is designed to prioritize user safety and ensure that the diagnostic process does not pose any risks to farmers or consumers. The mobile application will not require any hazardous materials or complex equipment, making it safe for use in the field. Furthermore, by enabling accurate disease identification, the system can help prevent the spread of diseases that may pose risks to both plant health and food safety. It is essential to ensure that any recommendations provided by the system are based on sound agricultural practices to avoid adverse effects on crop health.

2.3.4 Ethics

Ethical considerations for the plant disease detection project emphasize the importance of data privacy, accuracy, and inclusivity. The application is designed to be accessible for all farmers, regardless of their technological background or expertise, promoting equitable access to the tool. Accuracy and fairness are prioritized through rigorous validation of the deep learning model to prevent biases in disease identification, ensuring reliable results for all users.

2.3.5 Cost

The cost feasibility of the project includes several components, such as the development of the mobile application, data collection and labeling for training the CNN model, and ongoing maintenance and support. Initial investment may be required for technological infrastructure and software development. However, the long-term benefits of reduced crop losses and improved yields can outweigh these costs. Additionally, potential partnerships with agricultural organizations or government programs could provide funding support and resources to lower the overall cost for farmers.

2.3.6 Type

For this application, the ideal technology stack involves developing a machine learning model integrated with a Flask API for deployment. Python, as the primary programming language, is used for building and training the machine learning model, leveraging its rich ecosystem of libraries and tools.

2.3.7 Standards

The development of the proposed system will comply with relevant industry standards and best practices in both software development and agricultural technology. This includes adherence to data protection regulations to safeguard user information, as well as compliance with agricultural guidelines to ensure the accuracy and safety of disease diagnosis. Additionally, the system will be tested and validated according to established protocols in machine learning to guarantee its reliability and effectiveness in real-world applications.

2.4 SCOPE OF THE PROJECT

The scope of the plant disease detection project encompasses the development of a mobile application that empowers farmers to diagnose plant diseases accurately and efficiently using advanced deep learning techniques. This application will enable users to capture images of plant leaves and receive instant feedback on potential diseases, facilitating timely interventions. The project will involve creating a comprehensive dataset of labeled leaf images representing various plant species and diseases, essential for training the Convolutional Neural Network (CNN) model to classify up to 39 different plant disease categories based on visual symptoms. Usability will be a key focus, with an intuitive user interface designed for farmers with varying levels of technical expertise, alongside training and educational resources to enhance effective application usage. The project will also include evaluating system performance in real-world agricultural settings, allowing for continuous improvements based on user feedback and emerging plant diseases. Ultimately, the initiative aims to promote sustainable farming practices by reducing reliance on chemical treatments through accurate and timely disease identification, contributing to improved crop health and productivity.

2.5 SYSTEM CONFIGURATION

Software Requirements

1. Operating System-Windows/Mac/Linux
2. Jupyter Notebook
3. Spyder IDE
4. Visual Studio Code

Hardware Requirements

1. GPU integrated Laptop / Desktop
2. RAM : 8GB

3. LITERATURE OVERVIEW

1. **Mohanty, Sharada & Hughes, David & Salathé, Marcel. (2016). "Using Deep Learning for Image-Based Plant Disease Detection."**

   The study focuses on utilizing Convolutional Neural Networks (CNNs) to enhance the classification of plant diseases from leaf images, a critical area in agricultural technology. CNNs are particularly well-suited for image classification tasks due to their ability to automatically learn hierarchical features from visual data. The authors emphasize the necessity of large, well-annotated datasets for effectively training deep learning models, as this directly impacts the accuracy and reliability of disease identification. Their creation of a substantial dataset comprising over 54,000 images, meticulously collected and annotated to represent diverse plant species and diseases, is crucial for both training and validating the CNNs. This rigorous approach to data collection ensures that the model can generalize well and enhances its reliability when deployed in real-world agricultural settings. By addressing the quantity and quality of data, the researchers aim to mitigate common machine learning issues such as overfitting, ultimately demonstrating the effectiveness of CNNs in plant disease classification and setting a precedent for future research in the field.

2. **Ferentinos, Konstantinos. (2018). "Deep learning models for plant disease detection and diagnosis."**

   The research is about exploring the potential of deep learning algorithms in enhancing plant disease diagnosis. In this paper, convolutional neural network models were developed to perform plant disease detection and diagnosis using simple leaves images of healthy and diseased plants, through deep learning methodologies. Training of the models was performed with the use of an open database of 87,848 images, containing 25 different plants in a set of 58 distinct classes of [plant, disease] combinations, including healthy plants. The emphasis on data augmentation techniques suggests an understanding of the challenges associated with training models on limited datasets, particularly in agricultural contexts. By improving model robustness through augmentation, this study addresses a common issue in machine learning where overfitting can occur due to inadequate data.

3. **L. C. Ngugi, M. Abelwahab and M. Abo-Zahhad. (2020) "Recent advances in image processing techniques for automated leaf pest and disease recognition—A review."**
   In the paper the authors had presented a comprehensive review of recent research work done in plant disease recognition using IPTs, from the perspective of feature extracted based on hand-crafted or using deep learning techniques. And it is concluded that the deep learning techniques have superseded shallow classifiers trained using hand-crafted features. But they lacked some of the recent developments in terms of visualization techniques, and there is no mention of the early detection of the diseases and how to detect and classify plant diseases based on small samples.

4. **Khirade, Sachin D. and A. B. Patil. (2015) "Plant Disease Detection Using Image Processing."**
   The paper presents a method for identifying plant diseases through image processing techniques. The authors propose a system that involves acquiring leaf images, preprocessing them using techniques like noise removal and segmentation, and extracting key features such as texture and color. These features are then used in a classifier, like Support Vector Machines (SVM), to detect and classify diseases. The paper highlights challenges such as varying lighting conditions and complex backgrounds but concludes that the method offers a promising solution for early disease detection in agriculture, potentially improving crop health and yield.

5. **Gavhale, Ms & Gawande, Ujwalla. (2014). An Overview of the Research on Plant Leaves Disease detection using Image Processing Techniques.**
   In this research, authors outlined the importance of early detection for effective disease management in agriculture and survey different approaches, including image acquisition, preprocessing techniques. They discussed the challenges associated with detecting diseases due to varying environmental conditions and leaf characteristics. The paper emphasizes that image processing combined with machine learning provides a promising direction for accurate, automated plant disease detection, which can help improve agricultural productivity.

4. SYSTEM DESIGN

4.1 SYSTEM ARCHITECTURE

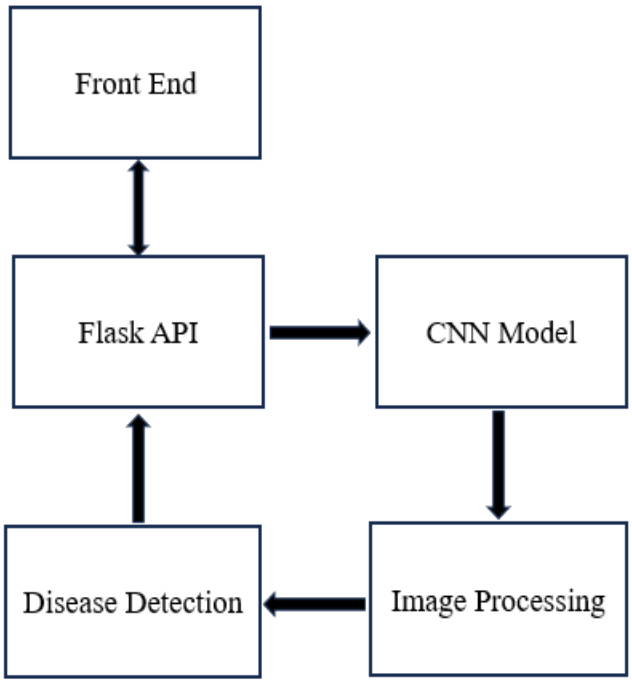

Fig 4.1.1 System Architecture

The Plant Disease Detection System is designed to provide an efficient and user-friendly approach for identifying diseases in plants using image recognition and machine learning techniques. The architecture consists of several interconnected components that facilitate the flow of data from the user to the model and back.

The process begins at the Front End, where users interact with the system through a user-friendly interface. This interface allows users to upload images of plants, specifically focusing on leaves or other plant parts that may exhibit signs of disease. Once the image is uploaded, it is sent to the Flask API, which serves as the intermediary between the user interface and the backend processing components. The Flask API handles incoming requests and routes them appropriately for further processing.

Upon receiving the image, the CNN Model is invoked to perform initial processing. This model is designed to analyze the image using deep learning techniques, particularly convolutional neural networks (CNNs). Before the model can make accurate predictions, the uploaded image undergoes Image Processing. This step includes techniques such as resizing, normalization, and other transformations to enhance the image quality and prepare it for analysis. After the image has been processed, the CNN Model is utilized for Disease Detection. In this phase, the model analyzes the features extracted from the image to determine if the plant is healthy or infected, and if infected, it identifies the specific disease.

Once the disease detection process is completed, the results are sent back to the Flask API. This API then formats the information appropriately and forwards it to the Front End. Finally, the user receives the results on their interface, which may include the classification of the plant's health status and recommendations for treatment or further actions.

This architecture allows for a seamless and interactive experience, enabling users to quickly diagnose plant diseases and take timely action. The modular nature of the system also facilitates maintenance and improvements, allowing for updates to the model or processing techniques without disrupting the overall functionality. By integrating these components effectively, the Plant Disease Detection System serves as a valuable tool for farmers, researchers, and plant enthusiasts.

4.1.1 Module Description

4.1.1.1 User Interface (UI) Module

The User Interface (UI) Module is the front-end component of the Plant Disease Detection System, providing a seamless and user-friendly experience. It allows users to upload images of plant leaves effortlessly and interact with the system intuitively. The interface is designed with a focus on usability, ensuring that users can easily navigate through the application. Once the analysis is complete, the UI presents the results clearly, showing whether the leaf is diseased, healthy, or if no leaf was detected. The output is made easily interpretable to enhance the user's understanding, helping them quickly grasp the health status of the plant. Additionally, the UI interacts smoothly with the Flask API, enabling efficient submission of images and retrieval of results.

4.3.1.2 Application Server Module (Flask API)

The Application Server Module is built using the Flask framework and serves as the backbone of the system, managing all server-side operations. This module handles HTTP requests from the UI and directs them to the appropriate handlers. It ensures smooth communication between the front end and the machine learning model by managing image uploads, processing requests, and routing responses back to the UI. The Flask API plays a crucial role in the system, ensuring that the user's interactions are processed efficiently and that the model's predictions are returned accurately.

4.3.1.3 Machine Learning Module

The Machine Learning Module encompasses the core functionality of the Convolutional Neural Network (CNN) used for disease detection in plant leaves. This module handles essential tasks such as data preprocessing, including image resizing and normalization, to prepare the inputs for model analysis. It is responsible for designing and training the CNN model, ensuring that it can accurately classify leaf images. After the model processes an image, it predicts whether the leaf is healthy, diseased, or absent. Additionally, this module evaluates the model's performance over time, continuously monitoring accuracy and ensuring that the disease detection system maintains a high level of reliability.

As a team, we developed the Plant Disease Detection System by carefully integrating the UI, Flask API, and CNN modules. Each member contributed to the various phases of development, from designing the user interface to optimizing the machine learning model for better accuracy. By working collaboratively, we ensured that each module functioned cohesively, creating a robust system capable of identifying plant diseases effectively.

We also focused on continuous evaluation and improvement of the model's performance, ensuring that the system remained reliable over time. Regular testing and performance tracking allowed us to fine-tune the system, addressing any challenges that arose during development. This collaborative approach ensured a high standard of quality and effectiveness throughout the project.

4.2 UML DIAGRAMS

1. USECASE DIAGRAM

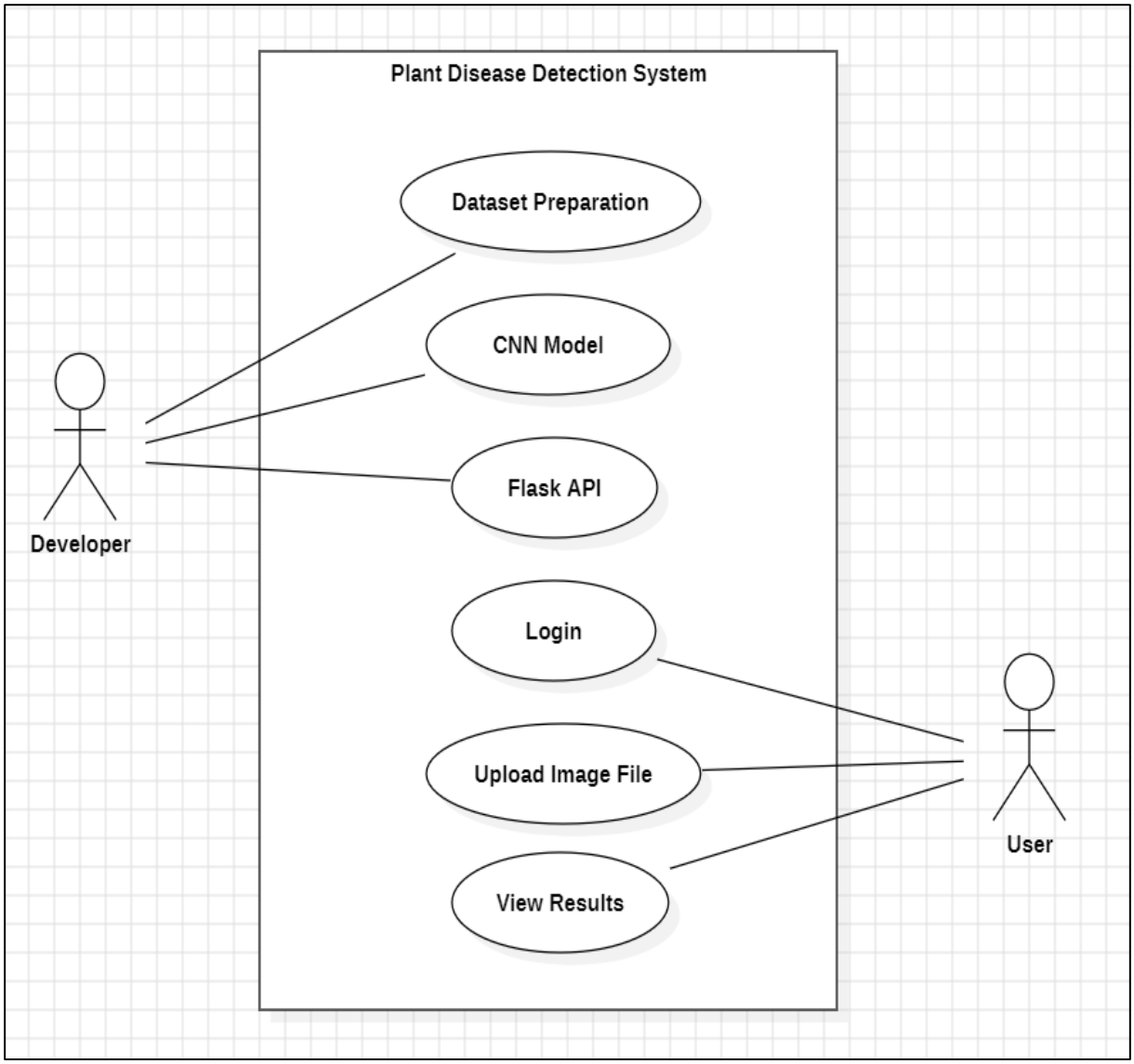

Fig 4.2.1 Use Case Diagram

2. ACTIVITY DIAGRAM

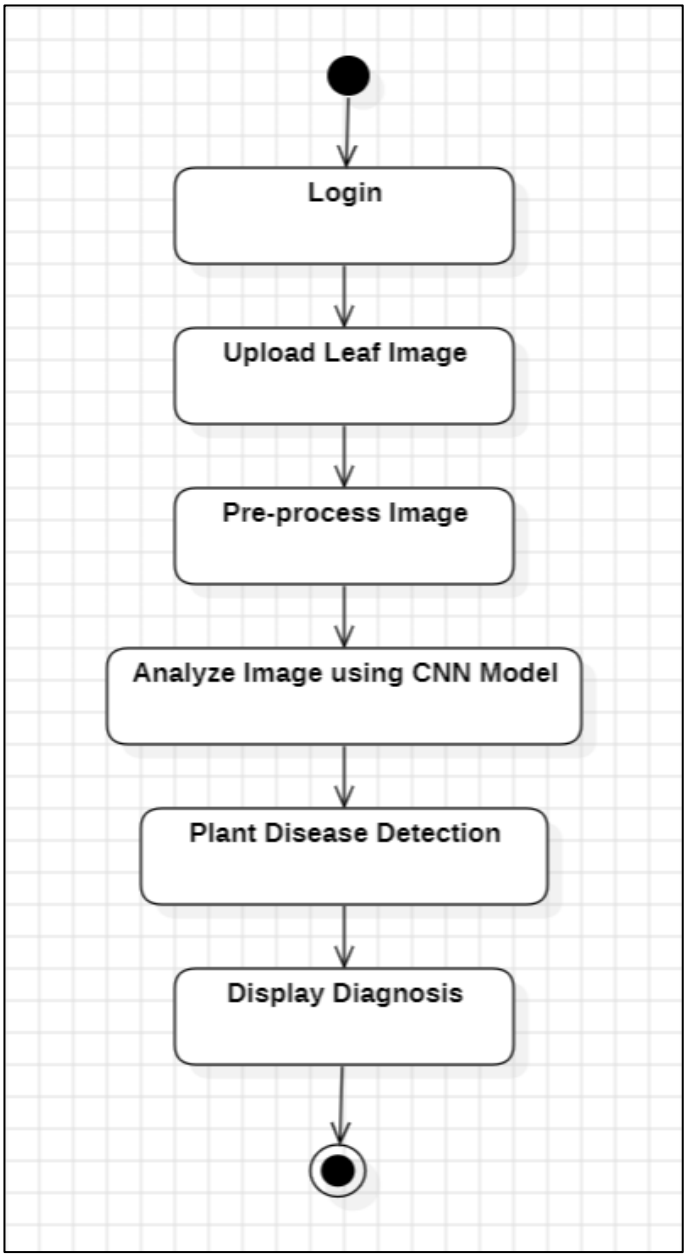

Fig 4.2.2 Activity Diagram

3. CLASS DIAGRAM

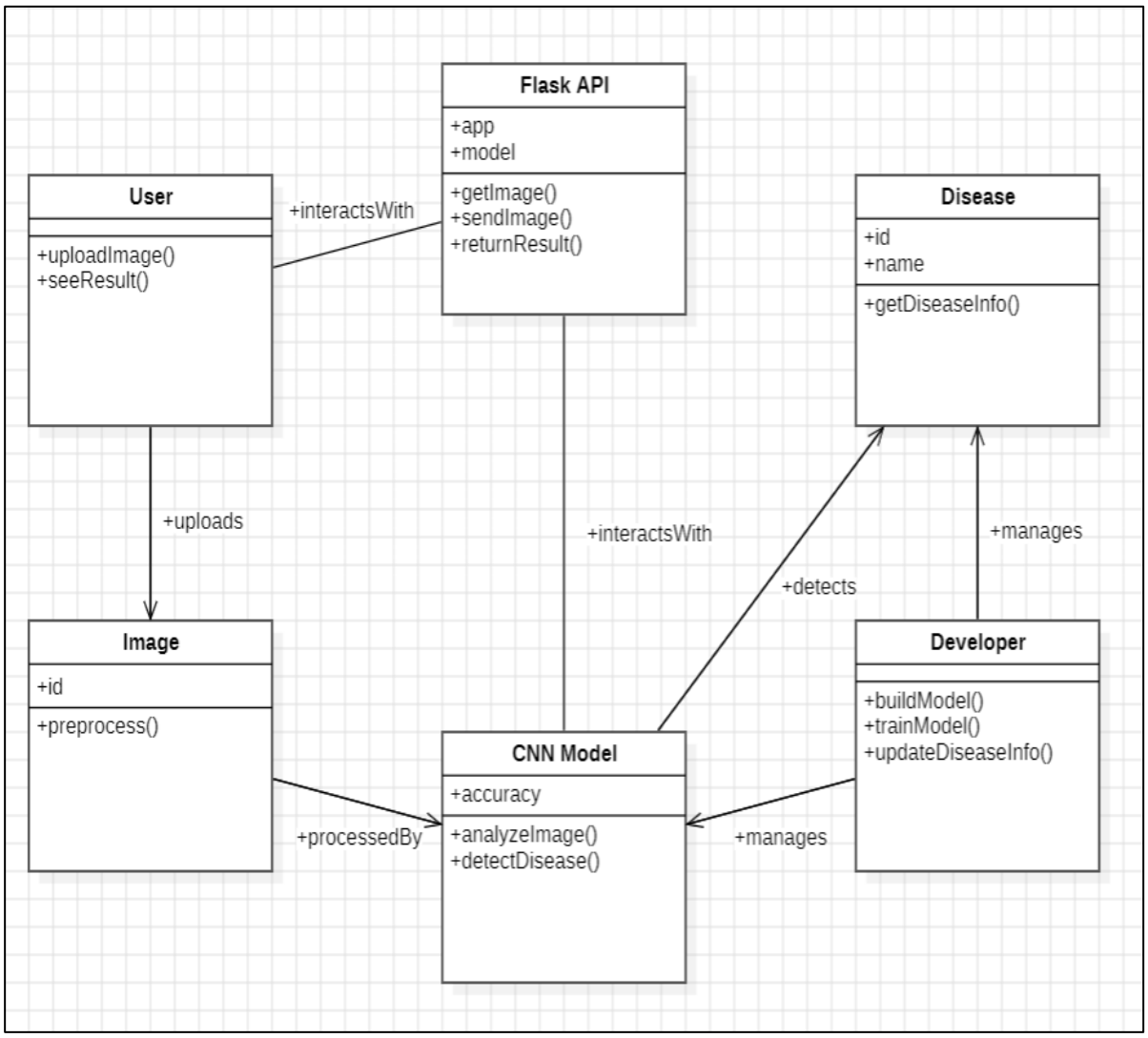

Fig 4.2.3 Class Diagram

4. SEQUENCE DIAGRAM

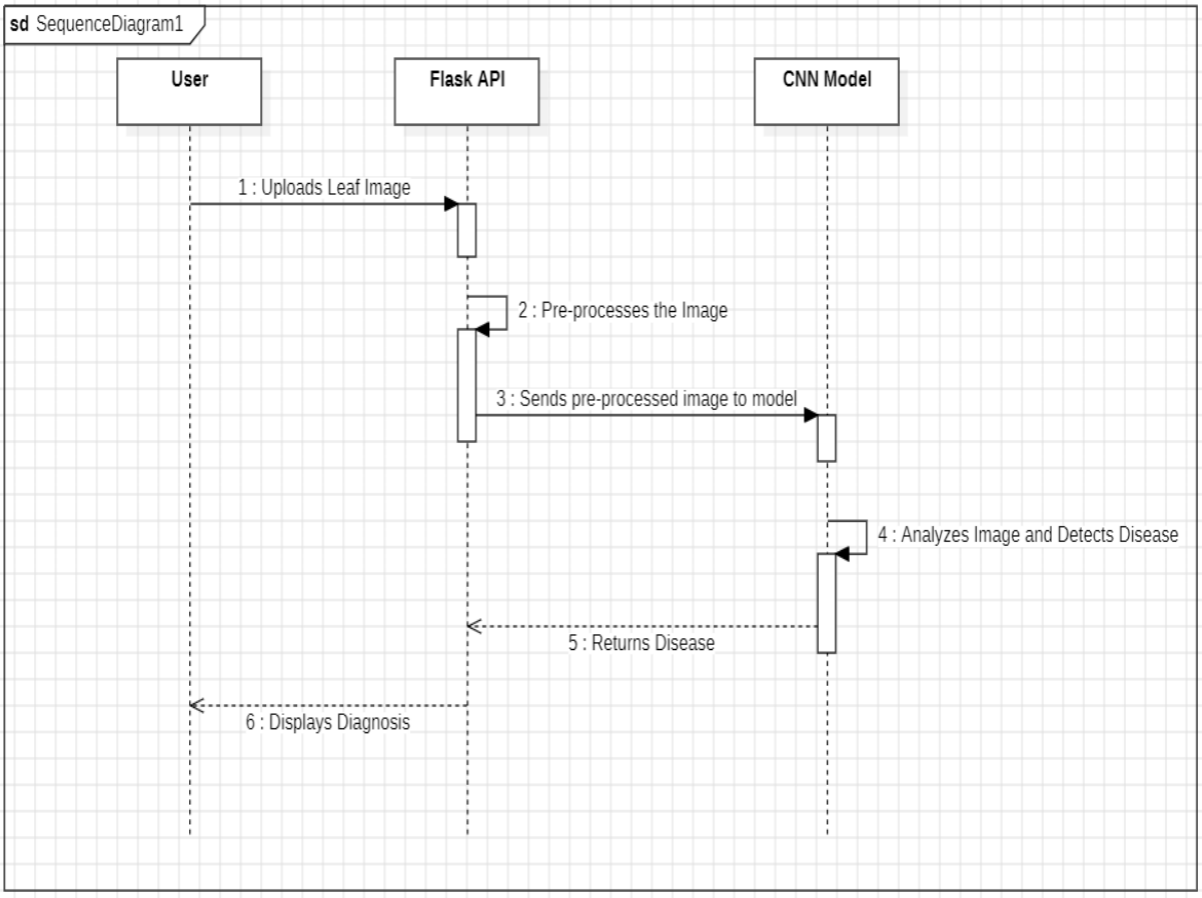

Fig 4.2.4 Sequence Diagram

4.3 SYSTEM DESIGN

4.3.1 Modular Design

4.3.1.1 User Interface (UI) Module

This module provides the front-end interface of the application, allowing users to interact with the system. It includes pages for uploading plant leaf images, viewing disease detection results, and navigating the application.

- A user-friendly interface for users to upload images of plant leaves for analysis.
- A section to present the output of the disease detection, including messages such as the type of disease detected, whether the leaf is healthy, or if no leaf was found.
- Interacts with the Flask API to submit uploaded images and retrieve results.

4.3.1.2 Application Server Module (Flask API)

The core server-side component that manages the application logic, processes requests from the UI, and communicates with the machine learning model.

- Manages incoming HTTP requests from the UI, routing them to the appropriate handlers for processing.
- Contains the logic for processing image uploads, and managing responses based on predictions from the model.
- Facilitates communication between the Flask API and the backend CNN model, sending uploaded images for analysis and receiving predictions.

4.3.1.3 Machine Learning Module

This module handles the training, deployment, and inference of the Convolutional Neural Network (CNN) model used for detecting diseases in plant leaves.

- Involves data preprocessing steps (e.g., image resizing, normalization), design and training of the CNN model, and optimization of model parameters.
- Responsible for classifying uploaded leaf images by analysing the input image and predicting whether it is diseased, healthy, or has no leaf present.
- Implements methods for evaluating model accuracy, tracking performance metrics, and monitoring the model's effectiveness over time.

4.3.2 Database Design

4.3.2.1 Dataset Description

The dataset contains 39 different classes of plant leaves and background images. The dataset contains a total of 61,486 images of the following 39 classes of images:

S.No	Class
1	Apple_scab
2	Apple_black_rot
3	Apple_cedar_apple_rust
4	Apple_healthy
5	Background_without_leaves
6	Blueberry_healthy
7	Cherry_powdery_mildew
8	Cherry_healthy

9	Corn_gray_leaf_spot
10	Corn_common_rust
11	Corn_northern_leaf_blight
12	Corn_healthy
13	Grape_black_rot
14	Grape_black_measles
15	Grape_leaf_blight
16	Grape_healthy
17	Orange_haunglongbing
18	Peach_bacterial_spot
19	Peach_healthy
20	Pepper_bacterial_spot
21	Pepper_healthy
22	Potato_early_blight
23	Potato_healthy
24	Potato_late_blight
25	Raspberry_healthy
26	Soybean_healthy
27	Squash_powdery_mildew
28	Strawberry_healthy

29	Strawberry_leaf_scorch
30	Tomato_bacterial_spot
31	Tomato_early_blight
32	Tomato_healthy
33	Tomato_late_blight
34	Tomato_leaf_mold
35	Tomato_septoria_leaf_spot
36	Tomato_spider_mites_two-spotted_spider_mite
37	Tomato_target_spot
38	Tomato_mosaic_virus
39	Tomato_yellow_leaf_curl_virus

The dataset comprises two types of images: non-augmented and augmented. Non-augmented images are the original, authentic photographs of plant leaves, reflecting real-world conditions and providing a diverse representation of various species and their health statuses. These images serve as the foundational training data for the model. In contrast, augmented images are generated through various transformation techniques—such as rotation, flipping, scaling, cropping, color jittering, and noise addition—enhancing the dataset by creating additional training samples. This augmentation increases the dataset size, improves the model's generalization capabilities, and enhances its robustness against variability in real-world scenarios, ultimately leading to higher accuracy in detecting plant diseases. The combination of both image types ensures that the CNN model can effectively learn and adapt to the complexities of plant health assessment.

5. IMPLEMENTATION

5.1 IMPLEMENATION

5.1.1 Dataset

```
dataset

Dataset ImageFolder
    Number of datapoints: 61486
    Root Location: Dataset
    Transforms (if any): Compose(
                             Resize(size=255, interpolation=PIL.Image.BILINEAR)
                             CenterCrop(size=(224, 224))
                             ToTensor()
                         )
    Target Transforms (if any): None
```

Fig 5.1.1.1 Size of the dataset

5.1.2 Splitting od Dataset

```
print(f"length of train size :{validation}")
print(f"length of validation size :{split - validation}")
print(f"length of test size :{len(dataset)-validation}")

length of train size :36584
length of validation size :15679
length of test size :24902
```

Fig 5.1.2.1 Sizes of the train, test and validation datasets

```
train_indices, validation_indices, test_indices = (
    indices[:validation],
    indices[validation:split],
    indices[split:],
)
```

Fig 5.1.2.2 Splitting of the dataset

5.1.3 Summary of the CNN Model

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 32, 224, 224]             896
              ReLU-2         [-1, 32, 224, 224]               0
       BatchNorm2d-3         [-1, 32, 224, 224]              64
            Conv2d-4         [-1, 32, 224, 224]           9,248
              ReLU-5         [-1, 32, 224, 224]               0
       BatchNorm2d-6         [-1, 32, 224, 224]              64
         MaxPool2d-7         [-1, 32, 112, 112]               0
            Conv2d-8         [-1, 64, 112, 112]          18,496
              ReLU-9         [-1, 64, 112, 112]               0
      BatchNorm2d-10         [-1, 64, 112, 112]             128
           Conv2d-11         [-1, 64, 112, 112]          36,928
             ReLU-12         [-1, 64, 112, 112]               0
      BatchNorm2d-13         [-1, 64, 112, 112]             128
        MaxPool2d-14           [-1, 64, 56, 56]               0
           Conv2d-15          [-1, 128, 56, 56]          73,856
             ReLU-16          [-1, 128, 56, 56]               0
      BatchNorm2d-17          [-1, 128, 56, 56]             256
           Conv2d-18          [-1, 128, 56, 56]         147,584
             ReLU-19          [-1, 128, 56, 56]               0
      BatchNorm2d-20          [-1, 128, 56, 56]             256
        MaxPool2d-21          [-1, 128, 28, 28]               0
           Conv2d-22          [-1, 256, 28, 28]         295,168
             ReLU-23          [-1, 256, 28, 28]               0
      BatchNorm2d-24          [-1, 256, 28, 28]             512
           Conv2d-25          [-1, 256, 28, 28]         590,080
             ReLU-26          [-1, 256, 28, 28]               0
      BatchNorm2d-27          [-1, 256, 28, 28]             512
        MaxPool2d-28          [-1, 256, 14, 14]               0
          Dropout-29                [-1, 50176]               0
           Linear-30                 [-1, 1024]      51,381,248
             ReLU-31                 [-1, 1024]               0
          Dropout-32                 [-1, 1024]               0
           Linear-33                   [-1, 39]          39,975
================================================================
Total params: 52,595,399
Trainable params: 52,595,399
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 143.96
Params size (MB): 200.64
Estimated Total Size (MB): 345.17
----------------------------------------------------------------
```

Fig 5.1.3.1 Summary of the CNN Model

Observations:

- The depth and width of the model (number of filters and layers) suggest it is designed for a complex task, likely involving nuanced features that need to be captured.
- Given the high number of parameters relative to the potential amount of training data, there's a risk of overfitting, especially if the dataset is small or not diverse enough.
- The inclusion of batch normalization layers helps in stabilizing the training process, which is beneficial for convergence and can potentially allow for higher learning rates.
- ReLU activations help in introducing non-linearity, crucial for the model to learn complex relationships in the data.

5.1.4 Loss Plot

```
plt.plot(train_losses , label = 'train_loss')
plt.plot(validation_losses , label = 'validation_loss')
plt.xlabel('No of Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

Fig 5.1.4.1 Code for Loss Plot

Output:

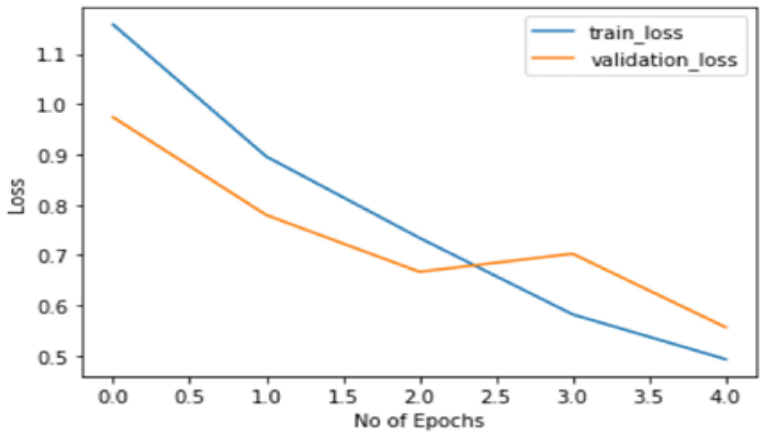

Fig 5.1.4.2 Loss Plot

Observation:

- The training loss (blue line) consistently decreases as the number of epochs increases, which indicates that the model is learning and improving its performance on the training data.
- The validation loss (orange line) decreases initially but starts to level off and even slightly increases after epoch 3. This could be a sign of slight overfitting, where the model's performance on the validation set is no longer improving even though it continues to improve on the training data.
- Up to the third epoch, the validation loss decreases at a similar rate as the training loss, showing good generalization. However, the slight increase in validation loss afterward suggests that the model might be starting to overfit to the training data, and further training may not significantly improve the model's performance on unseen data.

5.1.5 Accuracy of the CNN Model

```
train_acc = accuracy(train_loader)
test_acc = accuracy(test_loader)
validation_acc = accuracy(validation_loader)

print( f"Train Accuracy : {train_acc}\n
          Test Accuracy : {test_acc}\n
          Validation Accuracy : {validation_acc}" )

Train Accuracy : 96.7
Test Accuracy : 98.9
Validation Accuracy : 98.7
```

Fig 5.1.5.1 Accuracy of the CNN Model

Observations:

- A high training accuracy of 96.7% indicates that the model has effectively learned the patterns and features from the training data.
- A validation accuracy of 98.7% suggests that the model generalizes well to unseen data during training.
- A test accuracy of 98.9% serves as a final evaluation metric to assess the model's performance on entirely new data.

5.2 SAMPLE CODE

5.2.1 Front End – HTML Code

```
<html>
{% extends 'base.html' %}
{% block pagetitle %}
Search
{% endblock pagetitle %}
{% block body %}
<div>
  <div class="container">
    <div class="row mb-5 text-center text-white">
      <div class="col-lg-10 mx-auto">
    <link rel="stylesheet"
      href="https://fonts.googleapis.com/css?family=Pacifico&display=swap">
        <h1 class="display-4" style="padding-top: 2%;
            font-weight: 400;color: rgb(4, 54, 4);font-family: 'Pacifico',
            cursive;"><b>LeafSentry</b></h1>
        <p class="lead" style="font-weight: 500;color: black;font-style: italic;">
         Revolutionizing Plant Care with AI Precision</p>
      </div>
    </div>
    <div class="row ">
      <div class="col mx-auto">
      <div class="p-5 bg-white shadow rounded-lg" style="height: 95%;">
        <h5><b>Why is it important to detect plant diseases?</b></h5>
      <ul>
        <li>Plant diseases significantly impact the growth and health of plants.</li>
            <li>Early detection helps prevent the spread of diseases to other plants in the
              vicinity.</li>
            <li>Timely treatment reduces the need for aggressive and potentially harmful
```

```
                pesticides.</li>
            <li>Healthy plants contribute to higher agricultural yields and improved crop
                quality.</li>
            <li>Disease-resistant plant varieties can be identified and cultivated based on
                accurate diagnosis.</li>
            <li>Proper disease management minimizes economic losses for farmers and
                growers.</li>
            <li>Monitoring plant health enhances sustainable farming practices and
                environmental stewardship.</li>
        </ul>
    </div>
</div>
    <div class="col mx-auto">
        <div class="p-5 bg-white shadow rounded-lg" style="height: 95%;"><img
            src="{{ url_for('static', filename='images/plant.png') }}"
            height="300" alt="" width="300" class="d-block mx-auto mb-4 rounded-pill">
          <form action="/submit" method="POST" enctype="multipart/form-data">
            <div class="custom-file overflow-hidden mb-4">
                <input type="file" id="actual-btn" hidden name="image" />
                <label for="actual-btn">Choose File</label>
                <span id="file-chosen">No file chosen</span>
            </div>
            <h6 class="text-center mb-4 text-muted" style="font-weight: 500;
              color: black;font-style: italic;">
                Upload your plant's leaf image and witness the transformative power of AI.
            </h6>

            <center>
                <a class="mx-2"><button type="submit"
                    class="btn btn-outline-success">Detect</button></a>
```

```
                </center>
              </form>
            </div>
          </div>
          <div class="col mx-auto">
          <div class="p-5 bg-white shadow rounded-lg" style="height: 95%;">
            <h5><b>Prevent plant diseases with these steps:</b></h5>
            <ul type="square">
                <li>Keep garden tools clean to prevent disease spread.</li>
                <li>Use balanced fertilizers to maintain plant health.</li>
                <li>Check new plants for signs of disease before planting.</li>
                <li>Allow soil to warm up before planting to promote healthy growth.</li>
                <li>Rotate crops regularly to reduce disease buildup in soil.</li>
                <li>Trim plants to improve air circulation around foliage.</li>
                <li>Remove and dispose of diseased plant parts to prevent spread.</li>
            <li>Water plants carefully to avoid water-related diseases.</li>
            </ul>
            <a target="_blank" href="https://eos.com/blog/crop-diseases/" class="mx-2">
                <button type="button" class="btn btn-outline-success">
                  More Information</button>
            </a>
          </div>
        </div>
        </div>
    </div>
</div>
<script>
    const actualBtn = document.getElementById('actual-btn');
    const fileChosen = document.getElementById('file-chosen');
    actualBtn.addEventListener('change', function () {
```

```
        fileChosen.textContent = this.files[0].name
    })
</script>
{% endblock body %}
```

5.2.2 Flask API Code

```
import os
from flask import Flask, redirect, render_template, request
from PIL import Image
import torchvision.transforms.functional as TF
import CNN
import numpy as np
import torch
import pandas as pd

disease_info = pd.read_csv('disease_info.csv' , encoding='cp1252')
supplement_info = pd.read_csv('supplement_info.csv',encoding='cp1252')

model = CNN.CNN(39)
model.load_state_dict(torch.load("cnn_model.pt"))
model.eval()

def prediction(image_path):
    image = Image.open(image_path)
    image = image.resize((224, 224))
    input_data = TF.to_tensor(image)
    input_data = input_data.view((-1, 3, 224, 224))
    output = model(input_data)
    output = output.detach().numpy()
    index = np.argmax(output)
```

```
    return index

app = Flask(__name__)

@app.route('/')
def home_page():
    return render_template('home.html')

@app.route('/contact')
def contact():
    return render_template('contact-us.html')

@app.route('/index')
def ai_engine_page():
    return render_template('index.html')

@app.route('/mobile-device')
def mobile_device_detected_page():
    return render_template('mobile-device.html')

@app.route('/submit', methods=['GET', 'POST'])
def submit():
    if request.method == 'POST':
        image = request.files['image']
        filename = image.filename
        file_path = os.path.join('static/uploads', filename)
        image.save(file_path)
        print(file_path)
        pred = prediction(file_path)
        title = disease_info['disease_name'][pred]
```

```
        description =disease_info['description'][pred]
        prevent = disease_info['Possible Steps'][pred]
        image_url = disease_info['image_url'][pred]
        supplement_name = supplement_info['supplement name'][pred]
        supplement_image_url = supplement_info['supplement image'][pred]
        supplement_buy_link = supplement_info['buy link'][pred]
        return render_template('submit.html' , title = title , desc = description , prevent = prevent,
                                    image_url = image_url , pred = pred ,sname = supplement_name,
                                simage = supplement_image_url, buy_link = supplement_buy_link)

@app.route('/market', methods=['GET', 'POST'])
def market():
    return render_template('market.html', supplement_image = list
                                    (supplement_info['supplement image']),
                                    supplement_name = list(supplement_info['supplement name']),
                                    disease = list(disease_info['disease_name']),
                                    buy = list(supplement_info['buy link']))

if __name__ == '__main__':
    app.run(debug=True)
```

5.2.3 Backend Code

5.2.3.1 CNN Model

```
class CNN(nn.Module):
    def __init__(self, K):
        super(CNN, self).__init__()
        self.conv_layers = nn.Sequential(
            # conv1
            nn.Conv2d(in_channels=3, out_channels=32, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.BatchNorm2d(32),
            nn.Conv2d(in_channels=32, out_channels=32, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.BatchNorm2d(32),
            nn.MaxPool2d(2),
            # conv2
            nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.BatchNorm2d(64),
            nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.BatchNorm2d(64),
            nn.MaxPool2d(2),
            # conv3
            nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.BatchNorm2d(128),
            nn.Conv2d(in_channels=128, out_channels=128, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.BatchNorm2d(128),
```

```
            nn.MaxPool2d(2),
            # conv4
            nn.Conv2d(in_channels=128, out_channels=256, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.BatchNorm2d(256),
            nn.Conv2d(in_channels=256, out_channels=256, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.BatchNorm2d(256),
            nn.MaxPool2d(2),
        )

        self.dense_layers = nn.Sequential(
            nn.Dropout(0.4),
            nn.Linear(50176, 1024),
            nn.ReLU(),
            nn.Dropout(0.4),
            nn.Linear(1024, K),
        )

    def forward(self, X):
        out = self.conv_layers(X)

        # Flatten
        out = out.view(-1, 50176)

        # Fully connected
        out = self.dense_layers(out)

        return out
```

6. TESTING

6.1 SOFTWARE TESTING

Software testing is the process of evaluating a software application to ensure it functions as expected and is free from defects. It involves executing the software to identify bugs, check performance, security, and ensure it meets the specified requirements. It helps in identifying and rectifying defects, bugs, or issues.

6.1.1 Unit Testing

Unit Testing involves testing individual components or modules of a system to verify that each part functions correctly when tested independently from the rest of the system.

6.1.1.1. Model Prediction

The Convolutional Neural Network (CNN) model used for predicting plant diseases from leaf images is tested with various inputs to verify its accuracy.

- Correctly classified leaf images from the training dataset to confirm accuracy.
- Leaf images not present in the training dataset to test the model's ability to generalize and identify new cases.
- Edge cases, such as low-quality images, partially visible leaves, or different lighting conditions, to check the model’s robustness.

6.1.1.2. Frontend Components

Each user interface element, such as buttons, and image upload features, is tested to ensure they function as expected.

- The upload button triggers the image upload process and sends the image to the backend for disease prediction.
- Image preview functionality ensures users can view the uploaded leaf image before submitting for analysis.
- The results display properly shows the disease prediction and any relevant information without errors.

6.1.1.3. Backend Functions

The backend logic, including image processing and interactions with the CNN model, is tested.

- Handling of image upload and proper storage of the image on the server.
- Image preprocessing (such as resizing and colour adjustments) to ensure proper input is passed to the model.
- Interaction between the backend and the CNN model for accurate disease prediction and return of results to the frontend.

6.1.2. Integration Testing

Integration Testing focuses on verifying that different components of the plant disease detection system work together seamlessly.

6.1.2.1. Model Integration with Backend

The integration between the CNN model and the backend server is tested to ensure that uploaded leaf images are correctly processed and disease predictions are returned without issues.

- Checking the flow from image upload to prediction display, ensuring that the uploaded image is passed to the model and results are returned to the frontend.
- Ensuring the backend correctly handles model errors, timeouts, or failures, with proper error handling and fallback mechanisms.

6.1.2.2. Frontend and Backend Interaction

The communication between the frontend user interface and the backend server is tested.

- Verifying that user actions on the frontend (such as image uploads) trigger the correct backend processes for disease prediction.
- Ensuring that backend responses, including prediction results or error messages, are properly rendered and displayed on the frontend without glitches.

6.1.3. System Testing

System Testing involves testing the entire plant disease detection system as a whole to verify that it meets the specified requirements.

6.1.3.1. End-to-End Testing

The complete workflow is tested, starting from image upload through prediction to the display of results. This ensures that the system performs all intended functions correctly when used by an end-user.

- Testing the image upload process to confirm that users can successfully upload leaf images.
- Verifying the prediction process by ensuring that the model correctly analyses the uploaded images and returns accurate disease predictions.
- Checking the results display to ensure that users receive clear and informative feedback regarding the disease status of the plant.

6.1.3.2. Performance Testing

The system's performance is tested under various conditions to assess its response time, load handling, and scalability.

- Testing the prediction response time to ensure it is within acceptable limits, providing users with timely feedback after image submission.
- Testing the system's resource usage (CPU, memory, and bandwidth) during normal and peak conditions to ensure efficient operation and to identify potential bottlenecks that may affect performance.

6.2 TEST CASES

ID	Test Case Description	Test Steps	Expected Result	Test Result
TC-01	Verify that the homepage loads correctly and all elements are functional.	1. Navigate to the homepage URL. 2. Check that the page loads without errors. 3. Verify that the upload image button is visible and clickable. 4. Ensure any other elements are displayed correctly.	The homepage should load successfully with all elements visible and functional.	PASS
TC-02	Verify that the system accurately identifies and displays a disease detected in the uploaded leaf image.	1. Navigate to the image upload page. 2. Upload a valid leaf image known to have a specific disease. 3. Press the button to process the image. 4. Wait for the prediction output to be displayed.	The output should clearly indicate that the disease is detected, providing the name of the disease.	PASS
TC-03	Verify that the system correctly identifies and displays that the	1. Navigate to the image upload page. 2. Upload a valid leaf image known to be healthy.	The output should clearly indicate that the leaf is healthy.	PASS

	uploaded leaf is healthy.	3. Press the button to process the image. 4. Wait for the prediction output to be displayed.		
TC-04	Verify that the system accurately identifies when no leaf is present in the uploaded image.	1. Navigate to the image upload page. 2. Upload an image that does not contain any leaf. 3. Press the button to process the image. 4. Wait for the prediction output to be displayed.	The output should clearly indicate that no leaf is present in the image.	PASS

7. OUTPUTS

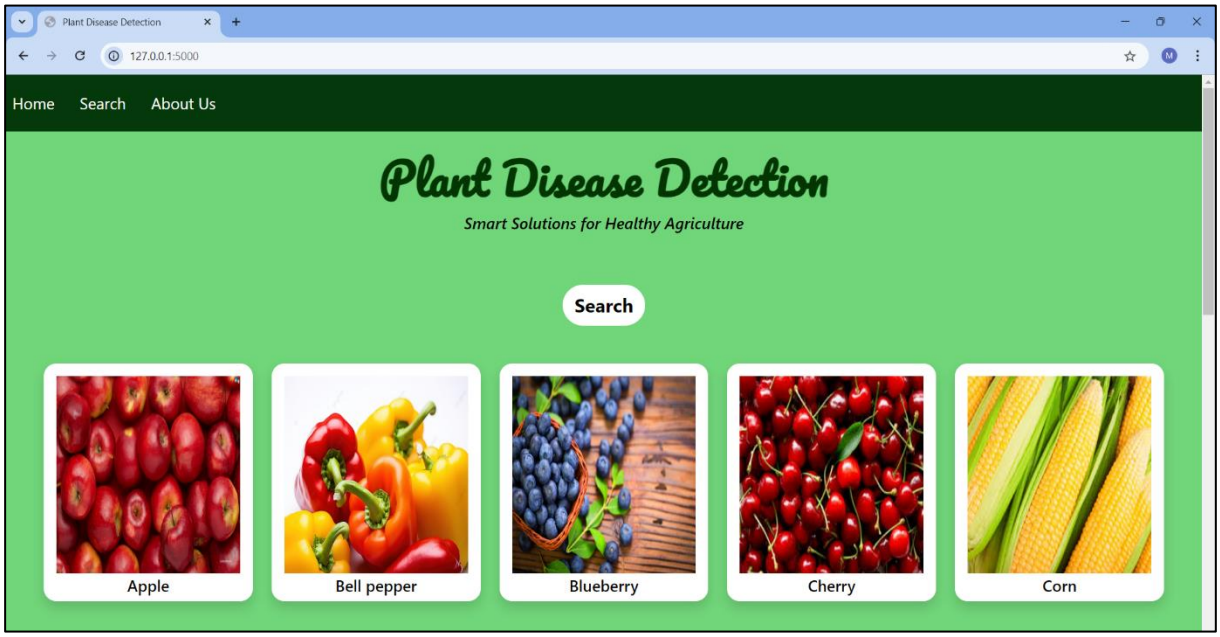

Fig 7.1 Home Page

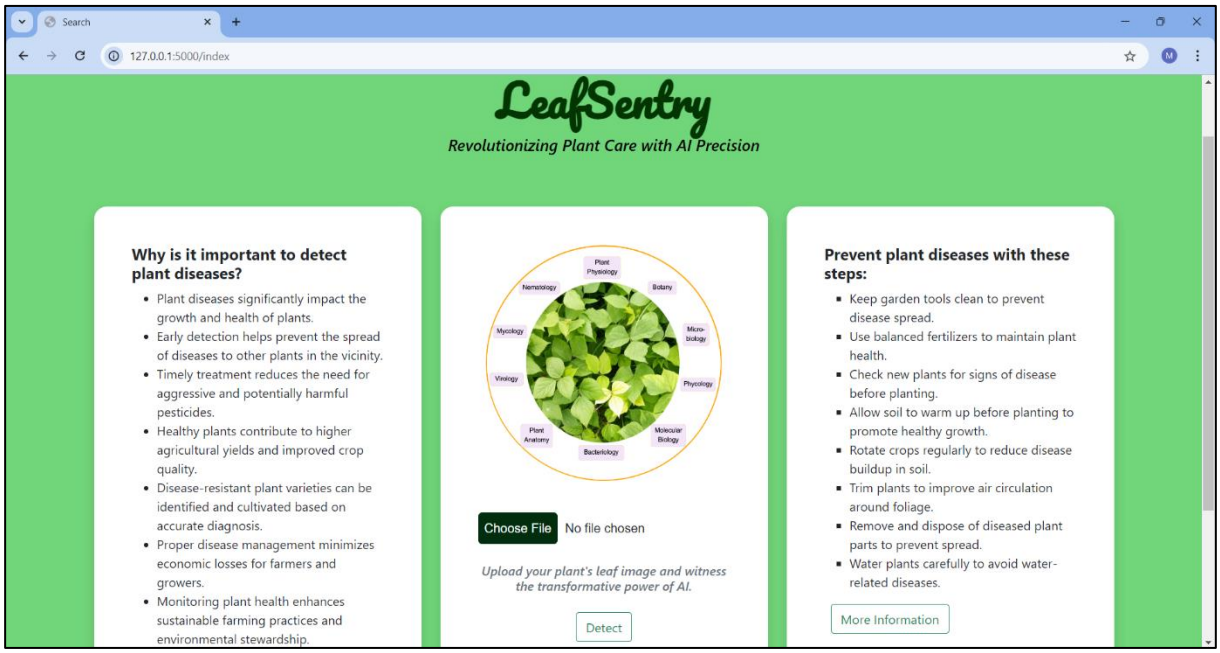

Fig 7.2 Search Page

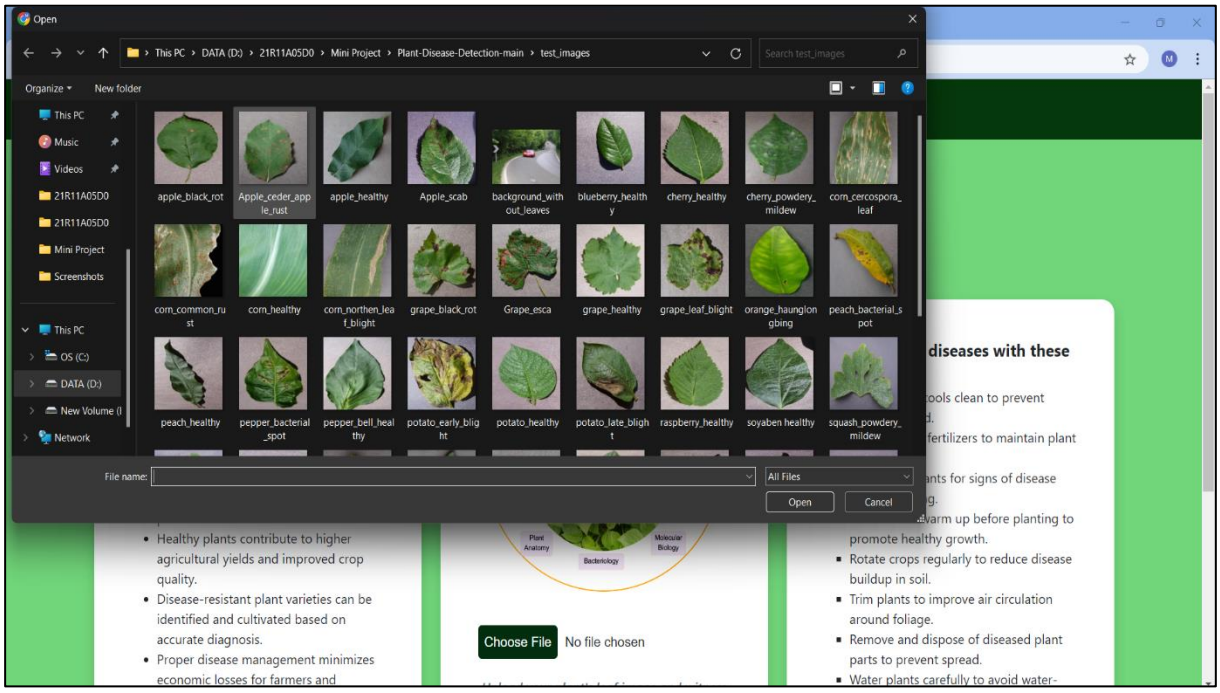

Fig 7.3 Selecting Image File

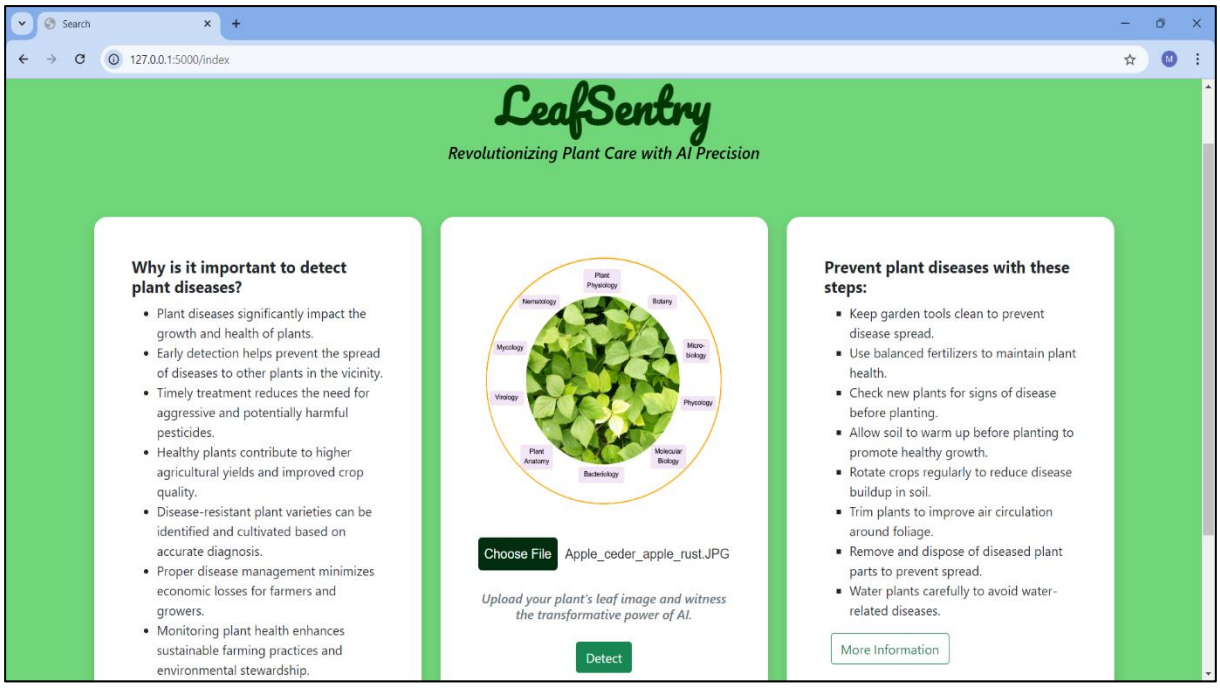

Fig 7.4 Successful Upload of Image File

Fig 7.5 Detection of Plant Disease

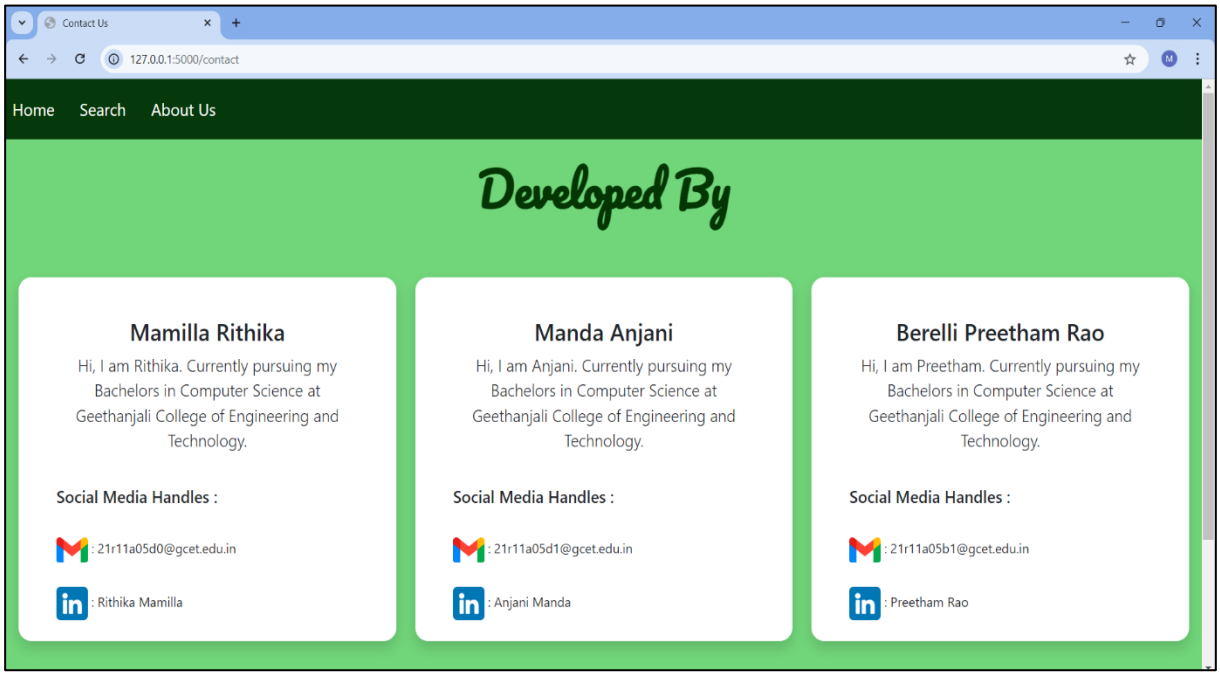

Fig 7.6 Team Page

8. CONCLUSION

8.1 CONCLUSION

In conclusion, the development of the plant disease detection system utilizing a Convolutional Neural Network (CNN) has yielded outstanding results, demonstrating a train accuracy of 96.7%, test accuracy of 98.9%, and validation accuracy of 98.7%. These impressive metrics indicate that the model is not only proficient in learning from the training data but also exhibits remarkable generalization capabilities when applied to unseen images. This level of accuracy positions the system as a reliable tool for accurately identifying plant health issues, which is crucial for farmers and agricultural professionals aiming to manage crops effectively.

The combination of both non-augmented and augmented images has significantly contributed to the model's robustness, allowing it to adapt to various real-world conditions and accurately classify plant leaves. The user-friendly interface and efficient backend processing further enhance the system's practicality, making it accessible to users without extensive technical expertise. Overall, this system offers a valuable solution for early detection of plant diseases, enabling proactive measures to safeguard crop health and optimize agricultural yields.

As the agricultural landscape continues to evolve, leveraging advanced technologies like this plant disease detection system can play a pivotal role in ensuring sustainable farming practices. With its high accuracy and user-centric design, the system is well-positioned to make a meaningful impact on crop management and food security.

8.2 FURTHER ENHANCEMENTS

To further improve the system's performance and usability, several enhancements can be implemented:

1. Expanding the dataset to include a wider variety of plant species and additional disease classes would provide the model with more comprehensive training data, enhancing its accuracy and generalization capabilities.
2. Implementing a user-friendly mobile application could allow users to easily capture and upload images directly from their smartphones, facilitating quicker and more accessible disease detection in the field.
3. Incorporating advanced techniques such as transfer learning, where a pre-trained model is fine-tuned on the plant disease dataset, could also yield significant improvements in accuracy while reducing training time.
4. Integrating a feedback mechanism for users to report prediction errors would provide valuable data for continuous model improvement.
5. Exploring real-time image processing and predictions could enhance the user experience, allowing for immediate diagnosis and recommendations, thereby contributing to more effective plant disease management and agricultural practices.

9. BIBLIOGRAPHY

9.1 Books References

1. Shanmugamani, R. (2018). Deep Learning for Computer Vision. Packt Publishing.
2. Chollet, F. (2017). Deep Learning with Python. Manning Publications.
3. Tiwari, P. (2020). Practical Machine Learning for Agriculture. BPB Publications.
4. Cheong, S. Y. (2020). Hands-On Image Generation with TensorFlow. Packt Publishing.
5. Singh, R. S., & Singh, U. S. (Eds.). (2007). Advances in Plant Disease Diagnosis. Daya Publishing House.

9.2 Websites References

1. Scikit-learn: Machine Learning in Python. https://scikit-learn.org
2. Flask Documentation. https://flask.palletsprojects.com
3. Pandas Documentation. https://pandas.pydata.org
4. TensorFlow Documentation. https://www.tensorflow.org
5. PyTorch Documentation. https://pytorch.org
6. Keras Documentation. https://keras.io
7. OpenCV Documentation. https://opencv.org

10. APPENDICES

10.1. APPENDIX A: SOFTWARE USED

1. Python: The primary programming language used for developing the machine learning model and handling backend functionality.
2. TensorFlow/Keras: Deep learning libraries utilized to build and train the Convolutional Neural Network (CNN) for disease detection in plant leaves.
3. Flask: A lightweight web framework used to create the web interface, manage user inputs, and serve model predictions.
4. HTML/CSS/JavaScript: Technologies employed for crafting the front-end interface, ensuring the website is interactive and responsive.

10.2. APPENDIX B: METHODOLOGIES USED

1. Convolutional Neural Network (CNN): A deep learning model specifically designed for image classification, applied here to detect plant diseases from leaf images.
2. Supervised Learning: The model was trained using a labelled dataset, where each plant leaf image was associated with its respective health condition or disease.
3. Web Development: The Flask framework was utilized to develop the web application, while HTML, CSS, and JavaScript were used to create the user-friendly interface.

10.3. APPENDIX C: MODULAR DESIGN

1. User Interface Module: Manages user interactions such as uploading plant leaf images and displaying the prediction results. Developed using HTML, CSS, and JavaScript.
2. Image Processing Module: Preprocesses the uploaded plant leaf images to ensure they are in the correct format for the CNN model to make predictions.
3. Prediction Module: Employs the trained CNN model to analyse the processed leaf images and predict whether the plant is diseased, healthy, or if no leaf is present.

10.4. APPENDIX D: INSTALLATION GUIDE

1. Download and Install Python

i. Installation instructions for setting up Python on your machine.

ii. Install Anaconda's latest version to access Spyder along Jupyter Notebook.

2. Install Flask and Required Libraries

i. Open a terminal/command prompt and run the following command to install Flask and other dependencies:

pip install flask tensorflow keras scikit-learn pandas numpy opencv-python

3. Accessing the Application

i. Instructions for accessing the web-based application through the browser (e.g., http://127.0.0.1:5000/).

ii. Open Spyder software run the Flask-API Code and the browser link will be displayed in the console.

11. PLAGIARISM REPORT

Fig 11.1 Plagiarism Report