# INDUSTRIAL ORIENTED MINI PROJECT

## Report

On

# AI-POWERED CROP DISEASE PREDICTION SYSTEM

Submitted in partial fulfilment of the requirements for the award of the degree of

**BACHELOR OF TECHNOLOGY**

**In**

**INFORMATION TECHNOLOGY**

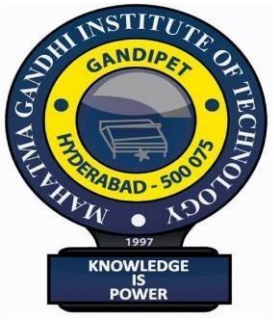
**By**

**Patlannagari Priteesh Reddy- 22261A1245**
**Vedant Malpani – 22261A1262**

Under the guidance of

**Dr. M .RudraKumar**

**Professor, Department of IT**




**DEPARTMENT OF INFORMATION TECHNOLOGY**

**MAHATMA GANDHI INSTITUTE OF TECHNOLOGY**

**(AUTONOMOUS)**

**(Affiliated to JNTUH, Hyderabad; Eight UG Programs Accredited by NBA; Accredited by NAAC with 'A++' Grade)**

**Gandipet, Hyderabad, Telangana, Chaitanya Bharati (P.O), Ranga Reddy District, Hyderabad– 500075, Telangana**

**2024-2025**

# CERTIFICATE

This is to certify that the **Project** entitled **AI-POWERED CROP DISEASE PREDICTION SYSTEM** submitted by **Patlannagari Priteesh Reddy(22261A1245), Vedant Malpani (22261A1262)** in partial fulfillment of the requirements for the Award of the Degree of Bachelor of Technology in Information Technology as specialization is a record of the bona fide work carried out under the supervision of **Dr. M. RudraKumar** , and this has not been submitted to any other University or Institute for the award of any degree or diploma.

**Internal Supervisor:**
**Dr. M . RudraKumar**
Professor
Dept. of IT

**IOMP Supervisor:**
**Dr. U. Chaitanya**
Assistant Professor
Dept. of IT

**EXTERNAL EXAMINAR**

**Dr. D. Vijaya Lakshmi**
Professor and HOD
Dept. of IT

# DECLARATION

We hear by declare that the **Industrial Oriented Mini Project** entitled **AI-POWERED CROP DISEASE PREDICTION SYSTEM** is an original and bona fide work carried out by us as a part of fulfilment of Bachelor of Technology in Information Technology, Mahatma Gandhi Institute of Technology, Hyderabad, under the guidance of **Dr. M. RudraKumar, Professor**, Department of IT, MGIT.

No part of the project work is copied from books /journals/ internet and wherever the portion is taken, the same has been duly referred in the text. The report is based on the project work done entirely by us and not copied from any other source.

**Patlannagari Priteesh Reddy- 22261A1245**

**Vedant Malpani- 22261A1262**

# ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of any task would be incomplete without introducing the people who made it possible and whose constant guidance and encouragement crowns all efforts with success. They have been a guiding light and source of inspiration towards the completion of the **Industrial Oriented Mini Project.**

We would like to express our sincere gratitude and indebtedness to our project guide **Dr. M. RudraKumar**, **Professor**, Dept. of IT, who has supported us throughout our project with immense patience and expertise.

We are also thankful to our honourable Principal of MGIT **Prof. G. Chandramohan Reddy** and **Dr. D. Vijaya Lakshmi**, Professor and HOD, Department of IT, for providing excellent infrastructure and a conducive atmosphere for completing this **Industrial Oriented Mini Project** successfully.

We are also extremely thankful to our Internal Supervisor **Dr U. Chaitanya**, Assistant Professor, Department of IT, and senior faculty **Mrs. B. Meenakshi** Department of IT for their valuable suggestions and guidance throughout the course of this project.

We convey our heartfelt thanks to the lab staff for allowing us to use the required equipment whenever needed.

Finally, we would like to take this opportunity to thank our families for their support all through the work. We sincerely acknowledge and thank all those who gave directly or indirectly their support for completion of this work.

**Patlannagari Priteesh Reddy- 22261A1245**

**Vedant Malpani- 22261A1262**

# ABSTRACT

The health of crops is vital for ensuring a stable and productive farming ecosystem, but diseases can spread quickly, causing significant crop damage and financial loss. Detecting and diagnosing plant diseases early is a challenging yet crucial task for farmers. This project aims to develop an **AI-powered Plant Disease Prediction System** that uses advanced machine learning and image processing techniques to predict potential plant diseases before they spread, allowing farmers to take timely action.

By leveraging AI algorithms and a robust dataset of plant images, the system analyzes plant health by identifying early signs of disease through visual patterns such as discoloration, spots, and deformations. Using **Convolutional Neural Networks (CNNs)** , the system not only detects visible symptoms but also predicts the likelihood of disease progression based on environmental factors like temperature, humidity, and soil conditions.

Farmers will be able to upload images of their crops through a **user-friendly web application**, where the system will predict possible diseases, assess the severity, and offer recommendations for preventative measures or treatments. This proactive approach helps farmers minimize the impact of plant diseases, reduce pesticide use, and optimize crop yields. The farmer will receive the message in his/her own mother tongue through any messaging app or through the user-friendly web application.

With this AI-powered prediction system, farmers can work smarter, not harder, by staying ahead of potential outbreaks, ensuring healthier crops, increasing their profits and contributing to more sustainable farming practices.

# TABLE OF CONTENTS


| Chapter No | Title | Page No |
|---|---|---|
| | **CERTIFICATE** | **i** |
| | **DECLARATION** | **ii** |
| | **ACKNOWLEDGEMENT** | **iii** |
| | **ABSTRACT** | **iv** |
| | **TABLE OF CONTENTS** | **v** |
| | **LIST OF FIGURES** | **vii** |
| | **LIST OF TABLES** | **viii** |
| **1** | **INTRODUCTION** | **1** |
| | 1.1 MOTIVATION | 1 |
| | 1.2 PROBLEM STATEMENT | 1 |
| | 1.3 EXISTING SYSTEM | 2 |
| | 1.3.1 LIMITATIONS | 2 |
| | 1.4 PROPOSED SYSTEM | 3 |
| | 1.4.1 ADVANTAGES | 3 |
| | 1.5 OBJECTIVES | 4 |
| | 1.6 HARDWARE AND SOFTWARE REQUIREMENTS | 4 |
| **2** | **LITERATURE SURVEY** | **6** |
| **3** | **ANALYSIS AND DESIGN** | **8** |
| | 3.1 MODULES | 8 |
| | 3.2 ARCHITECTURE | 9 |
| | 3.3 DESIGN USING UML | 10 |
| | 3.3.1 USE CASE DIAGRAM | 10 |
| | 3.3.2 CLASS DIAGRAM | 11 |
| | 3.3.3 ACTIVITY DIAGRAM | 14 |
| | 3.3.4 SEQUENCE DIAGRAM | 16 |
| | 3.3.5 COMPONENT DIAGRAM | 18 |
| | 3.3.6 DEPLOYMENT DIAGRAM | 20 |

| Chapter No | Title | Page No |
|---|---|---|
| | 3.4 METHODOLOGY | 21 |
| **4** | **CODE AND IMPLEMENTATION** | **25** |
| | 4.1 CODE | 25 |
| | 4.2 IMPLEMENTATION | 40 |
| | 4.2.1 DIRECTORY FOLDER | 40 |
| | 4.2.2 MODEL DEVELOPEMENT | 40 |
| | 4.2.3 INTEGRATION AND DEPLOYEMENT | 41 |
| | 4.2.4 USER INTERACTION FLOW | 41 |
| **5** | **TESTING** | 42 |
| | **5.1 INTRODUCTION TO TESTING** | 42 |
| | **5.2 TEST CASES** | 43 |
| **6** | **RESULTS** | **44** |
| **7** | **CONCLUSION AND FUTURE ENHANCEMENTS** | **47** |
| | 7.1 CONCLUSION | **47** |
| | 7.2 FUTURE ENHANCEMENTS | 48 |
| | **REFERENCES** | 49 |

# LIST OF FIGURES


Fig. 3.2.1 Architecture of Plant disease prediction system 9

Fig. 3.3.1.1 Use Case Diagram 10

Fig. 3.3.2.1 Class Diagram 12

Fig. 3.3.3.1 Activity Diagram 14

Fig. 3.3.4.1 Sequence Diagram 16

Fig. 3.3.5.1 Component Diagram 18

Fig 3.3.6.1 Deployment Diagram 20

Fig. 6.1 Performance Metrics 44

Fig. 6.2 Model Accuracy Graph 44

Fig. 6.3 Confusion Matrix 45

Fig. 6.4 Home Page 45

Fig 6.5 Image Uploading Page 46

Fig 6.6 Language options 46

Fig 6.7 Prediction Page 47

Fig 6.8 Notification 47

# LIST OF TABLES

Table 2.1 Literature Survey 7

Table 5.1 Test Cases of Plant disease predictor 43

# 1. INTRODUCTION

## 1.1 MOTIVATION

It is quite likely the most persistent challenge for farmers in the agricultural industry that timely plant disease identification and proper diagnosis become ever-present issues. Traditional disease identification methods are mostly dependent on expert visual examination, something that, though time-consuming, is still a luxury for most smallholder and marginal farmers. Such a delay in diagnosis can have far-reaching crop deterioration, economic losses, and increased food insecurity. Second, the majority of farmers might be lacking access to sophisticated agricultural advisory services, especially in rural or remote locations where expert advice is either limited or, in some cases, non-existent.

To overcome these challenges, the focus of our project is to utilize the application of artificial intelligence in the development of an image-based crop disease diagnostic system. One of the standout characteristics of our system is multilingual support, where disease reports and advice are transmitted in the farmer's mother tongue through SMS messages. This renders technology not an obstacle but an empowerment tool—offering farmers timely, understandable, and actionable information. The drive of this project is founded on the belief that all farmers, regardless of the degree of literacy or the location, should be given access to smart tools that protect their crops and secure their livelihoods.

## 1.2 PROBLEM STATEMENT

In addition to overcoming the language issue, the system is designed to overcome the issue of timely and accurate detection of crop diseases. The traditional method is through manual examination or advisory services, which are time-consuming, inconsistent, and generally out of reach for smallholder farmers. The solution employs computer vision and deep learning algorithms, in this case, Convolutional Neural Networks (CNNs), to detect plant diseases from images submitted by farmers. It provides real-time feedback with high accuracy, thereby preventing disease spread and enabling farmers to respond on time to prevent loss of crops and yield reduction.

In the majority of rural and regional farm environments, language is the primary barrier to access to agricultural technology. The majority of existing crop advisory and disease detection systems provide their output in English only or a few prominent languages only, making them unreadable and unimplementable by farmers who are largely conversant in

local or regional languages. To solve this problem, to make the solution available to more farmers, the proposed system is deployed with in-built multi-linguality. It allows farmers to receive crop disease reports, preventive recommendations, and treatment recommendations in their local languages, thereby making the solution more inclusive in nature and user-friendly. Adding automatic translation through APIs or pre-trained models, the system ensures that communication between artificial intelligence and farmers is transparent, understandable, and actionable.

## 1.3 EXISTING SYSTEM

Existing plant disease diagnosis systems are mainly based on image-based AI models or conventional database-based tools to aid farmers in disease identification for crops. They usually employ smartphone images or pre-existing disease databases to make predictions or recommendations. Some of the most popular platforms include Plantix, AgroDoctor, and CropIn with differing levels of intelligence, usability, and accessibility for farmers. Although these systems have helped to modernize agriculture, they tend to lack in several aspects such as multilingual support, real-time interaction, or customized feedback.

Plantix, by PEAT GmbH, utilizes smartphone photos to identify plant diseases, nutrient deficiencies, and pest attacks. It utilizes image-based artificial intelligence to diagnose symptoms and recommend solutions. It does not have wide multilingual support and necessitates stable internet connectivity, which restricts its use by farmers based in areas with poor connectivity. AgroDoctor provides a database of diseases and treatments but does not utilize artificial intelligence and hence is a static, non-real-time platform. CropIn's SmartFarm, by contrast, integrates artificial intelligence with satellite and Internet of Things data to facilitate holistic farm management but is custom-made for large-scale, enterprise-level farming and is not custom-made for individual or smallholder farmers.

### 1.3.1 Limitations of Existing Systems

- **Limited Multilingual Support**: Most of the systems offer core language support and do not cater to a broad linguistic range of farmers, especially in rural areas.
- **Connectivity Dependence**: Applications like Plantix rely on a connection to the internet, restricting their availability when offline or in low-network environments.

- **Absence of Real-Time Personalization:** AgroDoctor does not have AI-based real-time identification, whereas CropIn's business complexity renders it inaccessible to single users.
- **Unidirectional Feedback:** The majority of existing tools are not offering customized, actionable recommendations via personalized communication channels such as SMS, thereby reducing farmer interaction and prompt response.

## 1.4 PROPOSED SYSTEM

The proposed plant disease detection and prevention system leverages the power of artificial intelligence in terms of facilitating efficient and accurate plant disease identification through image processing. The backbone of the system is a Convolutional Neural Network (CNN) that is trained on a large dataset of images of crop leaves to detect and classify a variety of plant diseases. Farmers access a web-based portal to engage with the system, enabling them to submit images of crops that are suspected to be infected. The model processes these inputs to generate predictions in the form of whether the disease is of a specific type, the confidence level, and potential symptoms.

One of the most notable attributes of the system is that it supports multiple languages, expanding the access of agricultural practitioners from different linguistic backgrounds. Once the disease has been identified, the system generates a comprehensive diagnostic report that lists treatment recommendations particular to the disease identified. The report is then translated into the farmer's preferred language using a translation API, thereby promoting inclusivity and removal of linguistic barriers that would normally hinder the dissemination of specialized agricultural knowledge. The system further has a notification module that forwards the translated report to the farmer via SMS, thereby providing timely communication even where there is limited internet access.

### 1.4.1 Advantages

- **Real-Time Disease Detection**: The system facilitates rapid detection of farm diseases through AI-based analysis, thus eliminating diagnosis delays and enabling farmers to take remedial action on time.
- **Multilingual Communication**: The addition of automated translation services to the system enables the release of disease reports and treatment protocols in multiple

regional languages, which makes it significantly easier for farmers of different linguistic backgrounds to comprehend and access them.

- **Improved accessibility through SMS**: Utilization of SMS-based alerts guarantees that the less stable internet-connected individuals are not excluded and are still able to access timely information and assistance, hence rendering the system effective in rural and less connected regions.
- **Scalability and Adaptability**: The model's architecture is designed to be scalable to accommodate new crop and disease categories with little retraining required. Therefore, the system can easily be scaled up to accommodate a broader range of agricultural requirements.
- **Empowerment through Autonomy**: The proposed strategy reduces dependence on in-person farm advisement by offering farmers reliable, automated methods for disease diagnosis and management—thus inducing self-reliance and improving decision-making in crop cultivation.

## 1.5 OBJECTIVES

- To create an AI-based system that can efficiently identify crop diseases from images employing machine learning methods.
- With the aim to facilitate multilingual output to offer farmers disease diagnosis and treatment advice in their own languages.

## 1.6 HARDWARE AND SOFTWARE REQUIREMENTS

**Software Requirements**

- **Development Environment**: The project was implemented in Google Colab, which is a cloud-based Integrated Development Environment (IDE) with Python programming support and free GPU computing resources. The platform is particularly suitable for deep learning model training and testing due to its simplicity and pre-configured environment.
- **Primary Programming Language**: The application is written in Python, a high-level and general-purpose programming language that is extensively applied in artificial intelligence and data science. The primary libraries employed are

TensorFlow and Scikit-learn for modeling and NumPy and Pandas for preprocessing and data manipulation.

- **Server-Side Framework**: The system's backend is Flask-based, a lightweight Python web framework used to create and deploy the web application. It offers seamless integration between user interface and the AI model. While Django was also in the consideration list, the simplicity and flexibility of Flask made it more suitable to meet the project's requirements.

- **Frontend Technologies**: The frontend interface has been built using the following technologies to be user-friendly and responsive:

    1. HTML is employed for organizing the layout and contents of web pages.
    2. CSS – to add styles and visual effects.
    3. JavaScript – for interactive dynamics and user input management.
    4. Bootstrap – for a mobile-friendly and responsive layout.

**Hardware Requirements**

- **Operating System**: The platform is cross-compatible with market-leading operating systems, such as Windows 10/11 and Linux distributions, for local development and deployment.
- **Processor:** We recommend a multi-core Intel i5 processor or similar to enable real-time model inference, data loading optimization, and backend improvement.
- **RAM:** At least 8 GB RAM is required to support the computational needs of model training, prediction, and seamless operation of the Flask server.
- **Storage**: A minimum of 512 GB of disk space to store datasets, model files post-training (e.g., .h5 file format), log files, and other project resources.

# 2. LITERATURE SURVEY

Javidan et. alpresented a comprehensive review of feature engineering techniques used in conjunction with AI for plant disease identification. Their work emphasized the importance of symptom-based classification through image analysis, where features such as texture, color, and shape are extracted to train traditional ML and CNN models. While these approaches demonstrated acceptable accuracy, the need for large, well-labeled datasets and the computational cost of training complex models were major limitations. They also discussed the use of segmentation techniques like thresholding and clustering to improve image preprocessing, which is vital for effective disease classification. [1]

Balafas et al. proposed a comparative study on 18 state-of-the-art classification models and 5 object detection algorithms using the PlantDoc dataset. Their study found that while deep learning models such as ResNet50 and MobileNetv2 offered optimal trade-offs between accuracy and training time, they were prone to performance degradation in uncontrolled outdoor environments. Moreover, object detection models like YOLOv5 showed superior results for localizing disease-affected regions on leaves but required extensive GPU resources and tuning. [2]

Tirkey et al. introduced a real-time disease and insect detection framework tailored for the soybean crop using YOLOv5, InceptionV3, and custom CNN models. The system achieved high accuracy levels (above 97%) and incorporated pre-processing techniques like data augmentation and image segmentation. However, the approach struggled with environmental noise and varying image conditions such as lighting and background clutter, making consistent performance in field settings a challenge. [3]

Dolatabadian et al. offered a broad review of machine learning techniques used for image-based disease detection. The study highlighted that most current models are trained under controlled conditions and often fail in generalization when applied in real-world, dynamic environments. They also underscored the need for integration of contextual data such as weather, soil conditions, and crop management practices to improve the robustness of disease detection systems. [4]

Mishra et al. took a step toward smart agriculture by combining IoT sensors with a customized CNN model to develop a disease monitoring framework for millet crops. Their model leveraged temperature, humidity, and image data collected via sensors and processed through a Raspberry Pi-based system. The model was able to detect rust and blast diseases with over 98% accuracy and demonstrated scalability across other crop types. However, the

practical deployment of such IoT systems is limited by infrastructure constraints, sensor maintenance issues, and data transmission challenges in remote farming regions. [5]

Table 2.1 Literature Survey

| S. No | Author Names, Year | Journal / Conference Name & Publisher | Methodology / Algorithms Used | Merits | Demerits | Research Gaps |
|---|---|---|---|---|---|---|
| 1 | Javidan et al., 2024 | Elsevier – Smart Agricultural Technology | Feature Engineering, ML classifiers, Image Segmentation | Efficient feature extraction improves disease classification accuracy | Deep learning requires large annotated datasets | Need for more robust feature selection and hybrid approaches for small datasets |
| 2 | Balafas et al., 2023 | IEEE Access | CNNs (ResNet50, MobileNetV2), YOLOv5 | YOLOv5 best for object detection; ResNet50 balances speed and accuracy | Models fail in noisy, real-world conditions | Lack of robust models for uncontrolled environments; need for real-time lightweight models |
| 3 | Tirkey et al., 2023 | Elsevier – Smart Agricultural Technology | YOLOv5, InceptionV3, CNN with transfer learning | YOLOv5 offers 98.75% accuracy and real-time performance (53 fps) | Focused only on soybean, lacks generalizability | Require extension to multiple crops and multi-class disease detection |
| 4 | Dolatabadian et al., 2024 | Wiley – Plant Pathology | CNN, RNN, UAV-based imaging, ML fusion models | Integration of contextual data improves detection reliability | High implementation cost with advanced sensors | Limited real-world deployment and field validation; lack of integrated IoT-based disease response |
| 5 | Mishra et al., 2023 | Elsevier – Alexandria Engineering Journal | Customized CNN with IoT sensors, Raspberry Pi | 98.8% accuracy; scalable; real-time alerts for millet diseases | Disease focus limited to rust and blast in millet | Expandability to other crops and disease types; enhancement of cross-platform |

The reviewed literature highlights significant advancements in crop disease detection using CNNs, YOLO architectures, and IoT integration. While current models achieve high accuracy and real-time performance, challenges persist in generalizability, deployment in real-world noisy environments, and reliance on large annotated datasets. Future research should focus on lightweight, scalable, and multi-crop systems with robust field validation and hybrid data-driven approaches.

# 3. ANALYSIS AND DESIGN

The suggested crop disease diagnosis system through AI overcomes major agronomic issues by ensuring an automated process of disease identification from leaf images. The model uses a Convolutional Neural Network (CNN) to scan images loaded through a web portal and categorize them into disease classes with high accuracy. The system subsequently produces a report with the disease name, confidence level, symptoms, and suggested treatments.

What sets the system apart is that it is multilingual in its translation and SMS alert integration. The report is translated to the farmer's preferred language after its detection through the use of a translation API before it is sent directly via SMS, making it available even in areas with poor connectivity. The real-time artificial intelligence detection, localized message, and accessibility ensure that farmers receive timely and actionable information to defend their crops and increase productivity.

## 3.1 MODULES

The modules of this project are:

**1. Image Preprocessing Module**

- Enhances uploaded crop images to improve detection accuracy.
- Performs:
    - Resizing and normalization
    - Background noise reduction

**2. Disease Detection Module**

- Uses **Machine Learning / Deep Learning models (like CNN)** to analyze the image.
- Compares features with a trained dataset of healthy and diseased crop images.

**3. Multilingual Translation and Report generation Module**

- Converts diagnosis and treatment recommendations into the **user's selected regional language** .
- Generates the report based on the output in that particular language.
- Uses **translation APIs** or pre-trained language models.

**4. Notification & Recommendation Module**

Sends instant alerts and crop care suggestions via: SMS, App notification and email

## 3.2 ARCHITECTURE

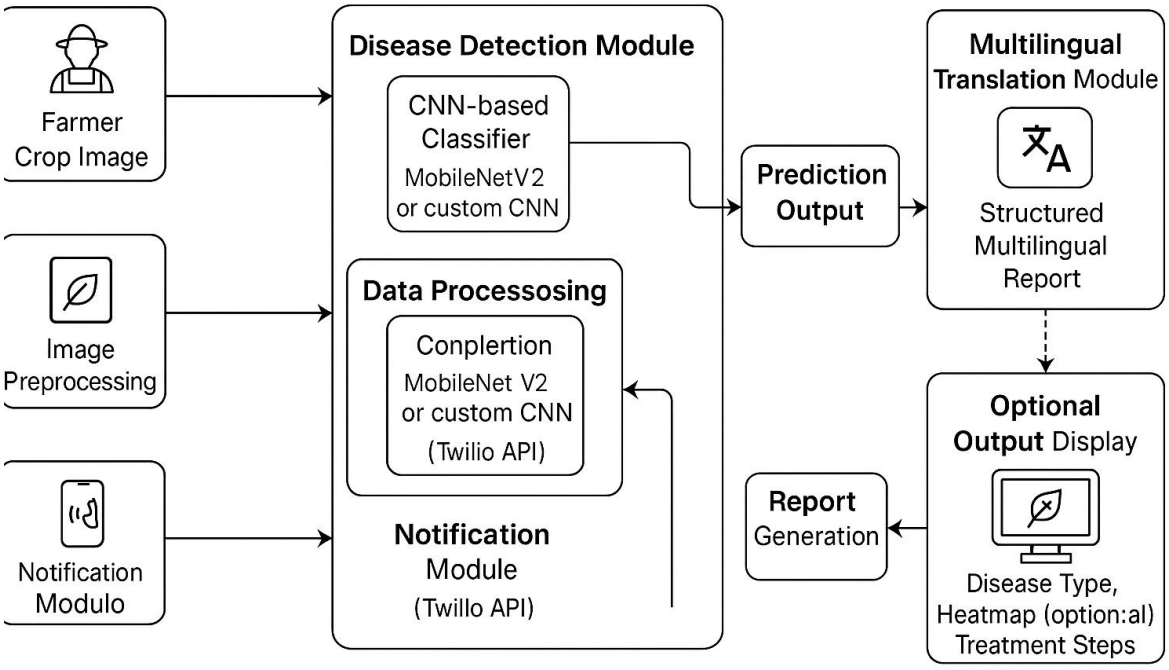



Fig. 3.2.1 Architecture of plant disease prediction system

The architecture of the system as shown in Fig. 3.2.1 is designed for efficient, accessible, and real-time crop disease detection, classification, and notification. The system begins with the farmer uploading a crop image through a web or mobile interface. This image is first routed through the Image Preprocessing Module, where it is resized, normalized, and converted into a format suitable for prediction.

Once pre-processed, the image is fed into the Disease Detection Module, where a trained CNN-based classifier (such as MobileNetV2 or a custom Convolutional Neural Network) analyzes the visual features to identify the disease. The Prediction Output—which includes the disease name and confidence score—is passed to the Multilingual Translation Module. This module utilizes a translation API to generate a structured report in the farmer's preferred language, making the results more accessible across diverse linguistic backgrounds.

Simultaneously, the result is sent to the Notification Module, which uses services like Twilio API to deliver the diagnosis and treatment information directly to the farmer via SMS. The system also supports an Optional Output Display, where the disease type, optional heatmap

visualization (if implemented), and suggested treatment steps are shown through a graphical interface. This architecture ensures that the disease identification process is not only fast and accurate but also user-friendly and inclusive for farmers in rural and multilingual regions..

## 3.3 DESIGN USING UML

### 3.3.1 USE CASE DIAGRAMS

A use case diagram is a visual representation that depicts the interactions between various actors and a system, capturing the ways in which users or external entities interact with the system to achieve specific goals. It is an essential tool in system analysis and design, often used in software engineering and business analysis. In a use case diagram, actors are entities external to the system that interact with it, and use cases are specific functionalities or features provided by the system as seen in Fig. 3.3.1.1. These interactions are represented by lines connecting actors to use cases. The diagram helps to illustrate the scope and functionality of a system, providing a high-level view of how users or external entities will interact with it.

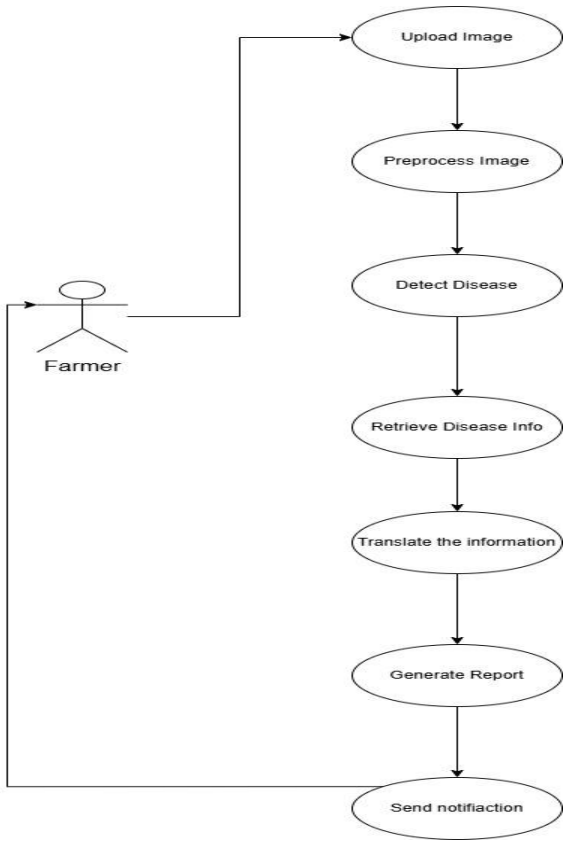



Fig. 3.3.1.1 Use Case Diagram of Crop disease Prediction System

**Actors:**

1. User (Farmer):

   Represents the user, typically a farmer, who will interact with the system to upload images of crops and receive disease diagnoses and recommendations.
2. System:

   The back-end application that performs the processing of the image, disease identification, report generation, content translation, and sending notifications.

**Use Cases**:

1. Upload Leaf Photo:

   The user uploads an image of the infected crop leaf via a web interface.
2. Preprocess Image:

   The system resizes and normalizes the uploaded image to prepare it for disease classification.
3. Recognize Illness:

   The pre-trained CNN model (e.g., MobileNetV2) classifies the image as a disease class or healthy label.
4. Get Disease Information:

   The system takes relevant information, such as symptoms and recommended treatments, based on the expected disease.
5. Translate Disease Information

   The disease information obtained is further converted into the user's preferred language with a multilingual translation API. Produce Report A formatted report is then generated, with the disease name, level of confidence, symptoms, and treatment protocols. Send SMS Notification The report is delivered as an SMS alert through a service such as Twilio, and it is available even in areas of low connectivity.

### 3.3.2 CLASS DIAGRAM

A class diagram is a visual representation that models the static structure of a system, showcasing the system's classes, their attributes, methods (operations), and the relationships between them as seen in Fig. 3.4.2.1. It is a key tool in object-oriented design and is commonly used in software engineering to define the blueprint of a system.

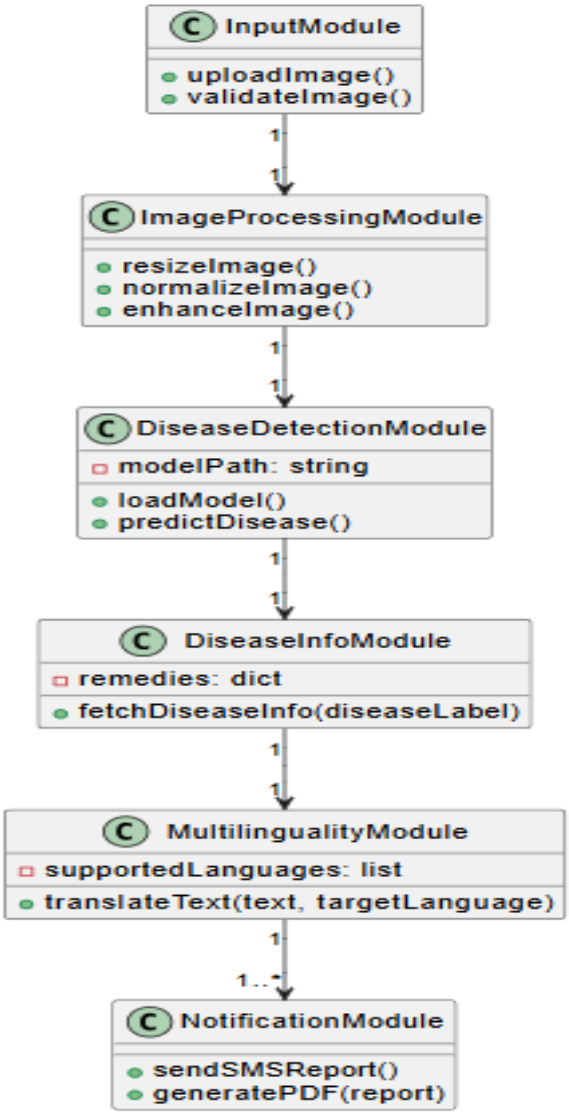



Fig. 3.3.2.1 Class Diagram of Crop Disease Prediction System

**Relationships**

1. **User→InputModule:**

   The user uploads or submits an image through the InputModule. The module handles image input and validation.

2. **InputModule→ImageProcessingModule:**

   The validated image is passed from the input module to the image processing module. This module performs operations such as resizing, normalization, and enhancement.

3. **ImageProcessingModule→DiseaseDetectionModule:**

   The processed image is sent to the DiseaseDetectionModule, which loads the AI model and performs disease prediction.

4. **DiseaseDetectionModule→DiseaseInfoModule:**
   Once a disease is predicted, its label is forwarded to the DiseaseInfoModule, which retrieves relevant information and remedy suggestions.
5. **DiseaseInfoModule→MultilingualityModule:**
   The disease information is passed to the multilingual module, which translates the content into the user's preferred language.
6. **MultilingualityModule→NotificationModule:**
   The translated report is sent to the notification module, which generates the output in PDF format and sends SMS notifications to the user.

**System Flow**

1. **Image Submission:**
   - The user uploads an image of a plant leaf via the **InputModule**.
   - The module checks if the image is valid and suitable for analysis.
2. **Preprocessing:**
   - The image is forwarded to the **ImageProcessingModule**.
   - It is resized, normalized, and enhanced for accurate detection.
3. **Prediction:**
   - The **DiseaseDetectionModule** uses a trained AI model to predict the disease.
   - It returns a disease label (e.g., "Leaf Blight").
4. **Disease Information Retrieval:**
   - The **DiseaseInfoModule** fetches remedy details and descriptions for the predicted disease.
5. **Language Translation:**
   - The **MultilingualityModule** translates the retrieved content into the user's selected language (e.g., Hindi, Tamil, etc.).
6. **Report Generation & Notification:**
   - The **NotificationModule** generates a multilingual PDF report and sends it via SMS to the user.
   - It may include plots or severity scores as part of the report.

### 3.3.3 ACTIVITY DIAGRAM

An Activity Diagram is a type of behavioral diagram used in Unified Modeling Language (UML) to represent the flow of control or data through the system as seen in Fig. 3.3.3.1. It focuses on the flow of activities and actions, capturing the sequence of steps in a particular process or workflow. Activity diagrams are commonly used to model business processes, workflows, or any sequential activities in a system.

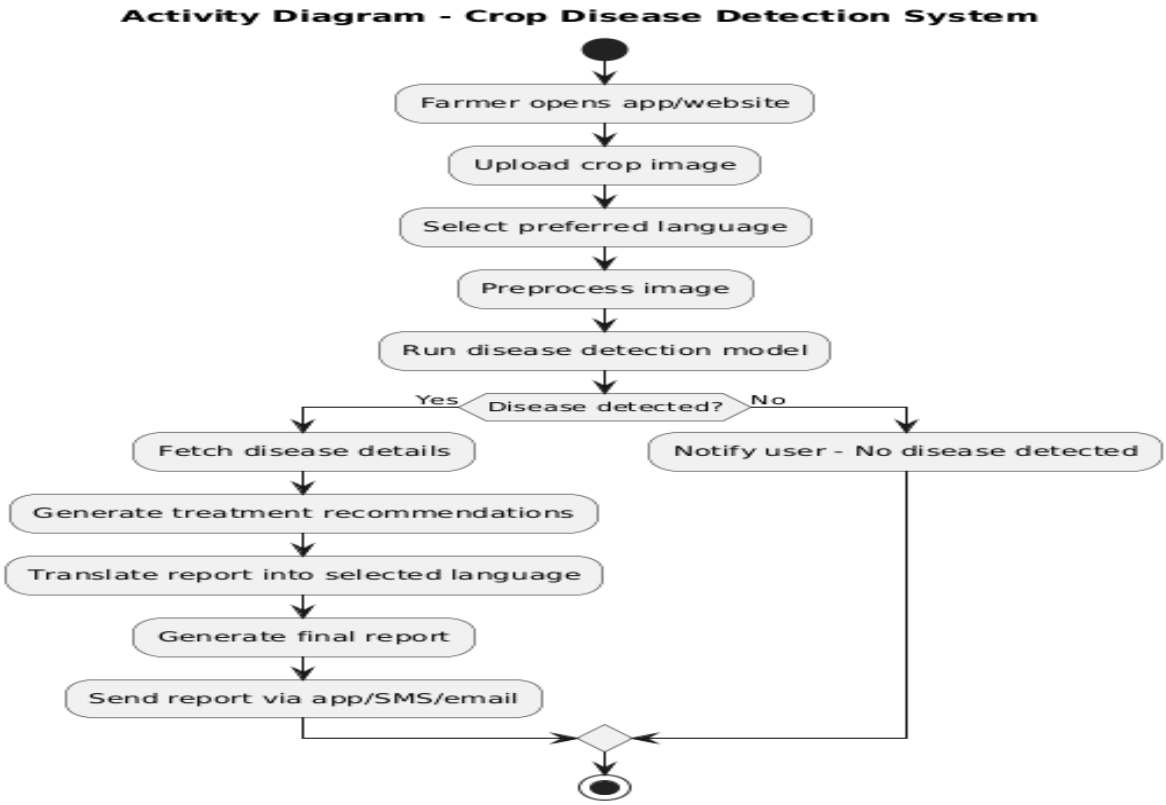



Fig. 3.3.3.1 Activity Diagram of Crop Disease Prediction System

**Flow Explanation**

**1. Open Application**

- The farmer begins by launching the mobile app or website designed for crop disease detection.

**2. Upload Crop Image**

- The user is prompted to upload an image of the affected crop leaf.
- The image is validated to ensure it meets requirements (e.g., format, size, visibility).

**3. Select Preferred Language**

- The user selects their preferred language for receiving results (e.g., English, Hindi, Tamil).
- This choice is stored and used for translating the final report.

**4. Preprocess Image**

- The uploaded image undergoes preprocessing, including:
    - Resizing to match model input dimensions.
    - Normalization and enhancement to improve prediction accuracy.

**5. Run Disease Detection Model**

- The preprocessed image is passed into a deep learning model.
- The model analyzes the image to detect the presence and type of disease.

**6. Disease Detection Decision**

- A decision point is reached:
    - **If a disease is detected**, the system continues to the next step.
    - **If no disease is detected**, the user is notified with a "No disease found" message, and the process ends.

**7. Fetch Disease Details**

- Based on the predicted disease label, relevant information such as:
    - Disease name
    - Description
    - Affected crops
    - Symptoms

      is fetched from the internal knowledge base or database.

**8. Generate Treatment Recommendations**

- The system compiles treatment guidelines and remedies:
    - Chemical treatments
    - Organic or cultural practices
    - Preventive measures

**9. Translate Report into Selected Language**

- The entire disease information and treatment guide is translated into the user's selected language for better accessibility and comprehension.

**10. Generate Final Report**

- A final report is compiled including:
    - Detected disease

- Treatment steps
- Translated content
- Images (optional)

- The report is formatted into PDF or visual output.

**11. Send Report via App/SMS/Email**

- The report is sent to the farmer through:
  - App interface
  - SMS notification
  - Email (if available)

**12. End of Workflow**

- The process ends. The user may return to upload another image or exit the application.

### 3.3.4 SEQUENCE DIAGRAM

A sequence diagram illustrates the flow of interactions between actors and system components over time as seen in Fig. 3.3.4.1, emphasizing the order in which messages are exchanged to achieve specific functionalities. Actors represent external entities that interact with the system, while lifelines depict the system components involved in the process. Messages are shown as arrows, indicating the flow of information or actions between these elements. By providing a step-by-step view of workflows, sequence diagrams help in understanding and designing the dynamic behaviour of a system.

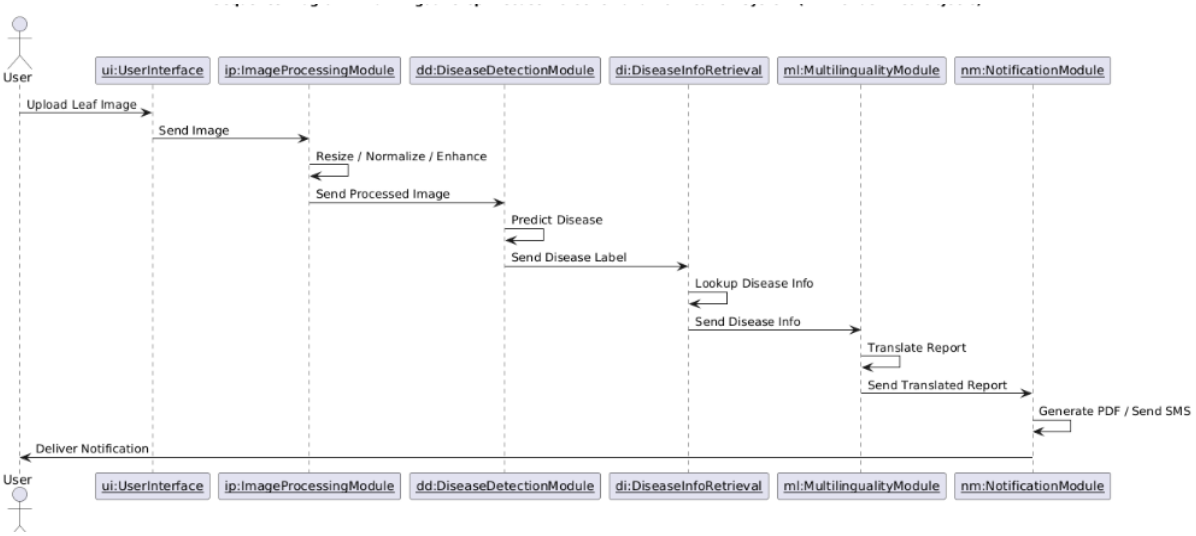



Fig. 3.3.4.1 Sequence Diagram of Crop Disease Prediction System

**Key Interactions and Relationships:**

**1. User and Interface:**

- The process begins with the **user uploading a crop leaf image** through the application's graphical interface.
- The image is sent to the **Image Processing Module** for analysis.

**2. Image Processing:**

- The module performs operations such as:
  - **Resizing** to standard dimensions.
  - **Normalization** for pixel intensity.
  - **Enhancement** to improve clarity.
- The **processed image** is then passed to the Disease Detection Module.

**3. Disease Detection:**

- This module receives the enhanced image and:
  - Uses an AI/ML model to **predict the disease**.
  - Outputs a **disease label** corresponding to the identified condition.
- The label is forwarded to the Disease Information module.

**4. Disease Information Retrieval:**

- Based on the predicted label, this module:
  - **Fetches detailed disease information** from a database or internal knowledge base.
  - Returns structured information such as:
    - Disease name
    - Description
    - Symptoms
    - Affected crops
    - Possible treatments

**5. Multilingual Translation:**

- The received disease information is **translated** into the user's **preferred language**, ensuring accessibility across regions.
- A **translated report** is created and passed on for final notification.

**6. Notification and Report Generation:**

- This module:
  - **Generates a PDF** of the report with visual charts and treatment guidelines.

- Sends the report via **SMS, app notification, or email** based on the user's contact preference.

**7. Notification to User:**

- The final notification or report is delivered to the user, concluding the sequence

### 3.3.5 COMPONENT DIAGRAM

A Component Diagram is a type of structural diagram used in software engineering to represent the components of a system and how they interact or depend on each other. It shows how the components (which could be software modules, subsystems, or other significant parts) are organized and connected within a system. In this diagram, each component encapsulates a set of related functionalities and interfaces as shown in Fig. 3.3.5.1.

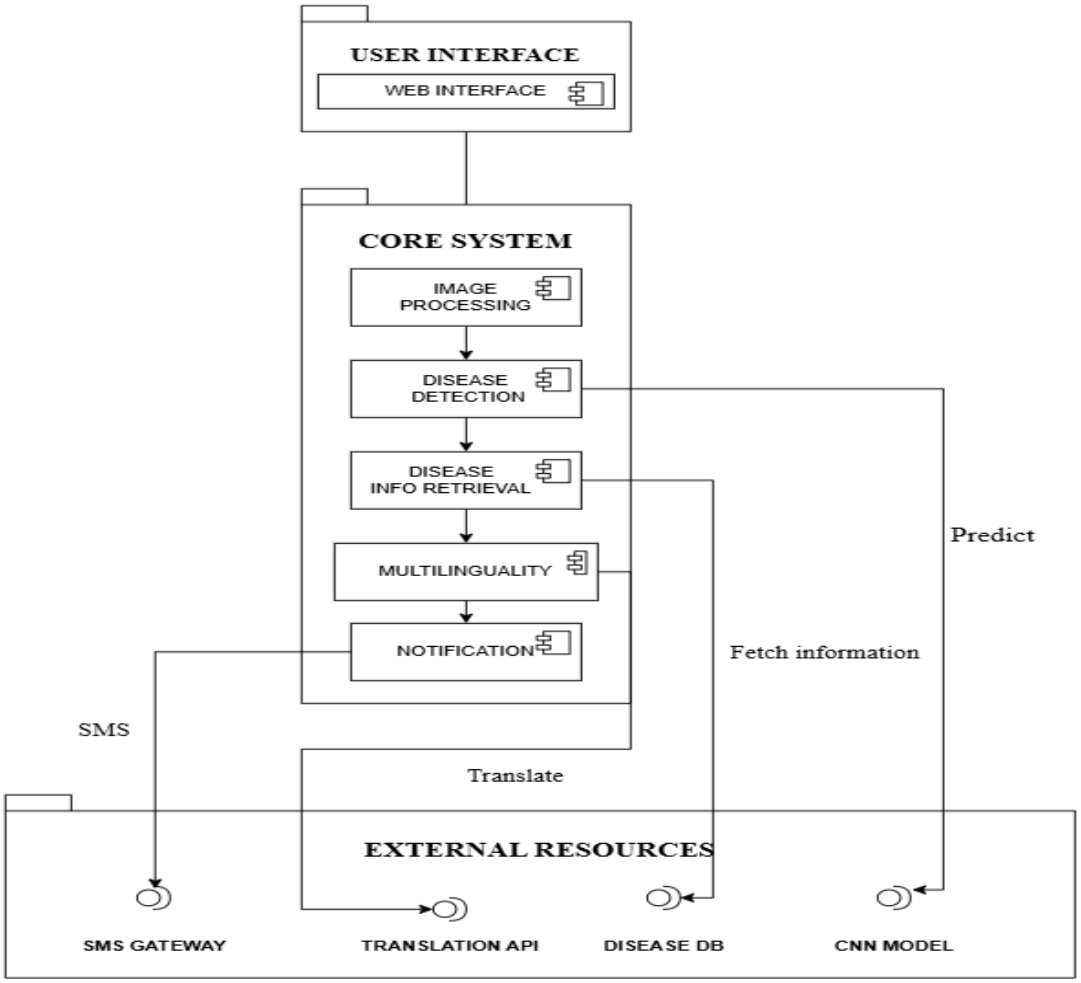



Fig. 3.3.5.1 Component Diagram of Crop Disease Prediction System

**Main Components:**

**1.User Interface:**

- This component allows users to interact with the system through a web interface.
- Users can upload leaf images and receive disease reports.
- It acts as the entry point for data submission and feedback reception.

**2.Image Processing Component:**

- This component processes the uploaded leaf image by performing enhancement operations such as resizing, normalization, and noise reduction.
- The processed image is then passed to the Disease Detection Component for analysis.

3.**Disease Detection Component:**

- This component uses the trained **CNNModel** to predict the type of disease present in the uploaded leaf image.
- The prediction result (disease label) is sent to the Disease Info Retrieval Component.

4.**Disease Info Retrieval Component:**

- This component fetches detailed information about the predicted disease using the **DiseaseDB**.
- The information includes symptoms, causes, and suggested treatments.

5.**Multilinguality Component:**

- This component translates the disease information into the user's preferred language using the **TranslationAPI**.
- It ensures that users can understand the disease report regardless of their native language.

6.**Notification Component:**

- This component generates a PDF or SMS-based report using the translated information.
- It then sends the report to the user via the **SMSGateway**.

7.**External Resources:**

- **SMSGateway:** Sends SMS notifications to the user containing the disease report.
- **TranslationAPI:** Provides translation services for multilingual support.
- **DiseaseDB:** Supplies disease-related data required by the system.
- **CNNModel:** A trained Convolutional Neural Network model used for disease prediction based on image input.

### 3.3.6 DEPLOYMENT DIAGRAM

The Deployment Diagram illustrates the physical deployment of software components across hardware nodes in the Prediction system. It captures the interaction between the components, highlighting how system components communicate and are distributed in a real-world deployment scenario.

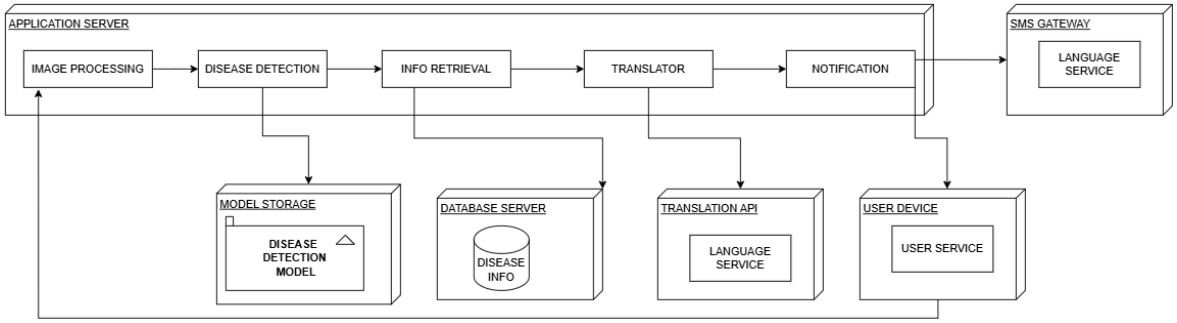



Fig 3.3.6.1 Deployment Diagram of Crop Disease Prediction System

The deployment diagram illustrates how the core components of the AI-powered multilingual plant disease detection system are distributed and interact across different nodes. The system is composed of the following major components:

**1. User Device**

- **Component**: Web/Mobile Interface (User Service)
- **Function**:

  The user device is used by farmers to upload images of crop leaves. It serves as the front-end interface where users interact with the system.
- **Communication**:

  Communicates with the application server over HTTP to send images and receive disease reports or translated recommendations via SMS.

**2. Application Server**

This is the main processing unit that hosts several modules:

**a. Image Processing**

- Preprocesses the uploaded leaf image for analysis (resizing, normalization, etc.).

**b. Disease Detection**

- Uses a trained deep learning model (stored in the **Model Storage**) to identify plant diseases from the processed image.

**c. Info Retrieval**

- Fetches relevant disease information and recommended treatments from the **Database Server**.

**d. Translator**

- Translates the disease report or treatment advice into the user's local language using a **Translation API**.

**e. Notification**

- Sends the translated disease diagnosis and recommendation to the farmer's mobile number through an integrated **SMS Gateway**.

**3. Model Storage**

- **Component**: Trained CNN model
- **Function**: Stores the AI model used for detecting the disease from the image.
- **Interaction**: Accessed by the Disease Detection module.

**4. Database Server**

- **Component**: Disease Information Repository
- **Function**: Stores text-based information about diseases, symptoms, preventive measures, and cures.
- **Interaction**: Accessed during the Info Retrieval phase.

**5. Translation API**

- **Component**: Language Service
- **Function**: Translates the disease information into the local language selected by the user.
- **Interaction**: Integrated with the Translator module and supports multilingual communication.

**6. SMS Gateway**

- **Component**: Language Service-Enabled Messaging System
- **Function**: Sends translated recommendations to the user's mobile number.
- **Interaction**: Final communication step post-translation.

## 3.4 METHODOLOGY

**Data Collection and Input Preparation:**

The dataset for training the model comprises thousands of high-resolution leaf images annotated with various plant diseases. These images were sourced from benchmark repositories such as the PlantVillage dataset, which includes both healthy and infected leaves across diverse crop types like tomato, apple, potato, grape, and more.

To ensure model generalization and prevent overfitting, the raw data undergoes multiple augmentation techniques, including:

- Random rotations and flips,
- Brightness and contrast adjustments,
- Zooming and cropping.

Each image is resized to 224×224 pixels and normalized (pixel values scaled to [0, 1]) to match the input requirements of MobileNetV2.

**Models Used**

**Convolution Neural Networks(CNNs):**

Convolutional Neural Networks (CNNs) are deep learning architectures that are naturally conducive to image recognition and classification. Their ability to learn spatial hierarchies of images makes them particularly well-suited to identify subtle patterns and features of plant disease.

**How it works**:

CNNs employ a sequence of convolutional layers, pooling layers, and fully connected layers to extract features from input images and comprehend them. Convolutional layers identify edges, textures, and patterns; pooling layers shrink dimensions without losing essential information; and fully connected layers utilize these features to classify.

**Why it's used:**

CNNs are particularly well-suited to identify crop disease since they are able to learn sophisticated visual patterns like discoloration, spots, or mold growth, which are important disease indicators. They are very accurate and don't need human features to be extracted, making them well-suited to autonomous farm use.

**Mobile Net V2 Architecture:**

**MobileNetV2** is a lightweight and computationally efficient CNN architecture optimized for mobile and embedded applications. Unlike traditional deep CNNs that are computationally intensive, MobileNetV2 uses a **depthwise separable convolution** strategy to drastically reduce the number of parameters and computational cost while maintaining high classification performance.

**How it works**:

MobileNetV2 introduces two key innovations:

- **Inverted Residuals with Linear Bottlenecks**: Instead of expanding and then shrinking features like typical residual blocks, it first reduces dimensionality through a bottleneck layer, applies a depthwise convolution, and then expands back to a higher dimension.
- **Depthwise Separable Convolutions**: These divide standard convolutions into two operations — depthwise convolution and pointwise convolution — which significantly reduces computation while preserving accuracy.

**Why it's used in the project**:

Given the agricultural context and the need for real-time, resource-efficient predictions on potentially low-power devices (e.g., smartphones or Raspberry Pi-based systems), MobileNetV2 provides an ideal trade-off between speed, memory efficiency, and classification accuracy. Its lightweight structure allows deployment in cloud or edge-based systems for farmers without requiring expensive hardware.

This combination of a powerful CNN backbone (MobileNetV2) and an integrated multilingual, farmer-friendly reporting system ensures the model is both **technically robust** and **practically accessible** for deployment in real-world agricultural settings.

## ALGORITHM:

Step 1: Data Preprocessing

1.1 Load raw images from dataset

1.2 Apply image augmentation:

- Random rotation, flipping

- Zoom and crop

- Brightness/contrast adjustment

1.3 Resize images to 224x224 pixels

1.4 Normalize pixel values to range [0, 1]

Step 2: Model Architecture (MobileNetV2)

2.1 Load MobileNetV2 base model (with pre-trained ImageNet weights)

2.2 Freeze base layers (optional during initial training)

2.3 Add custom classification head:

- GlobalAveragePooling2D

- Dense layer with ReLU activation

- Output layer with Softmax activation (for N classes)

Step 3: Training

3.1 Compile the model with:

- Loss function: categorical_crossentropy

- Optimizer: Adam

- Metrics: accuracy

3.2 Train model on training data with validation split

- Use early stopping to avoid overfitting

- Save best-performing model weights

Step 4: Evaluation

4.1 Evaluate model on validation/test data

4.2 Generate classification report and confusion matrix

Step 5: Deployment

5.1 Integrate model into farmer-friendly interface

5.2 Use multilingual reporting module

5.3 Send prediction and advice via SMS (using Twilio API)

# 4. CODE AND IMPLEMENTATION

## 4.1 CODE

**app.py**

```
import os
from PIL import Image
import numpy as np
import tensorflow as tf
from flask import Flask, render_template, request, redirect, send_file, url_for
from googletrans import Translator
from twilio.rest import Client
from reportlab.pdfgen import canvas
from io import BytesIO
import cv2
from werkzeug.utils import secure_filename
from datetime import datetime
from reportlab.lib.pagesizes import A4
from reportlab.lib.utils import ImageReader

from dotenv import load_dotenv
load_dotenv()

app = Flask(__name__)
translator = Translator()

# Configuration
app.config['UPLOAD_FOLDER'] = 'static'
app.config['ALLOWED_EXTENSIONS'] = {'png', 'jpg', 'jpeg'}

# Load the model
model = tf.keras.models.load_model("crop_disease.h5")

# Class labels
classes = ['Apple___Apple_scab', 'Apple___Black_rot', 'Apple___Cedar_apple_r',
'Apple___healthy',
        'Blueberry___healthy', 'Cherry_(including_sour)___Powdery_mildew',
        'Cherry_(including_sour)___healthy', 'Corn_(maize)___Cercospora_leaf_spot
Gray_leaf_spot',
        'Corn_(maize)___Common_rust_', 'Corn_(maize)___Northern_Leaf_Blight',
'Corn_(maize)___healthy',
        'Grape___Black_rot', 'Grape___Esca_(Black_Measles)',
'Grape___Leaf_blight_(Isariopsis_Leaf_Spot)',
        'Grape___healthy', 'Orange___Haunglongbing_(Citrus_greening)',
'Peach___Bacterial_spot',
        'Peach___healthy', 'Pepper,_bell___Bacterial_spot', 'Pepper,_bell___healthy',
'Potato___Early_blight',
        'Potato___Late_blight', 'Potato___healthy', 'Raspberry___healthy', 'Soybean___healthy',
        'Squash___Powdery_mildew', 'Strawberry___Leaf_scorch', 'Strawberry___healthy',
        'Tomato___Bacterial_spot', 'Tomato___Early_blight', 'Tomato___Late_blight',
'Tomato___Leaf_Mold',
```

```
        'Tomato___Septoria_leaf_spot', 'Tomato___Spider_mites Two-spotted_spider_mite',
'Tomato___Target_Spot',
        'Tomato___Tomato_Yellow_Leaf_Curl_Virus', 'Tomato___Tomato_mosaic_virus',
'Tomato___healthy']

# Suggestions
suggestions = {
 "Apple___Apple_scab": "Remove fallen leaves. Apply sulfur fungicide weekly. Plant resistant
varieties like 'Liberty'.",
"Apple___Black_rot": "Prune infected branches. Apply captan fungicide. Remove all
mummified fruit.",
"Apple___Cedar_apple_r": "Remove nearby junipers. Apply myclobutanil at bud break. Plant
resistant varieties.",
"Apple___healthy": "Maintain proper pruning. Monitor regularly. Keep area weed-
free.","Blueberry___healthy": "Maintain acidic soil (pH 4.5-5.5). Use pine bark mulch. Irrigate
at base.",
"Cherry_(including_sour)___Powdery_mildew": "Apply potassium bicarbonate. Improve air
circulation. Avoid wetting leaves.",
"Cherry_(including_sour)___healthy": "Prune for sunlight penetration. Monitor for pests.
Maintain consistent moisture.",
"Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot": "Rotate with soybeans. Apply
azoxystrobin. Space plants properly.",
"Corn_(maize)___Common_rust_": "Plant early-maturing varieties. Spray chlorothalonil.
Remove volunteer plants.",
"Corn_(maize)___Northern_Leaf_Blight": "Till crop residues. Use resistant hybrids. Apply
fungicide at tasseling.",
"Corn_(maize)___healthy": "Ensure proper nitrogen levels. Monitor for pests. Rotate crops
annually.","Grape___Black_rot": "Remove mummified berries. Apply mancozeb weekly.
Train vines vertically.",
"Grape___Esca_(Black_Measles)": "Disinfect pruning tools. Avoid severe pruning. Apply
borax to cuts.",
"Grape___Leaf_blight_(Isariopsis_Leaf_Spot)": "Apply copper fungicide. Remove infected
leaves. Keep vineyard clean.",
"Grape___healthy": "Prune properly. Monitor for pests. Maintain balanced
nutrition.","Orange___Haunglongbing_(Citrus_greening)": "Remove infected trees. Control
psyllids with imidacloprid. Use reflective mulch.","Peach___Bacterial_spot": "Apply copper
sprays in dormancy. Avoid overhead irrigation. Plant resistant varieties.",
"Peach___healthy": "Prune for airflow. Monitor for borers. Maintain consistent
watering.","Pepper,_bell___Bacterial_spot": "Use disease-free seeds. Apply copper sprays.
Space plants adequately.",
"Pepper,_bell___healthy": "Rotate crops. Avoid wetting foliage. Monitor for
aphids.","Potato___Early_blight": "Remove infected leaves. Apply chlorothalonil. Maintain
nitrogen levels.",
"Potato___Late_blight": "Destroy infected plants. Apply mancozeb preventatively. Hill
potatoes properly.",
"Potato___healthy": "Use certified seed potatoes. Monitor soil moisture. Rotate
crops.","Raspberry___healthy": "Prune old canes. Maintain weed-free rows. Provide trellising
support.","Soybean___healthy": "Rotate with corn. Monitor for aphids. Maintain proper pH
(6.0-6.8).","Squash___Powdery_mildew": "Apply neem oil. Plant resistant varieties. Water at
soil level.","Strawberry___Leaf_scorch": "Remove old leaves after harvest. Apply copper
fungicide. Improve drainage.",
"Strawberry___healthy": "Renovate beds annually. Use straw mulch. Monitor for spider
mites.","Tomato___Bacterial_spot": "Treat seeds with hot water. Apply copper+mancozeb.
Avoid working with wet plants.",
```

```
"Tomato___Leaf_Mold": "Reduce humidity. Apply potassium bicarbonate. Space plants properly.",
"Tomato___Septoria_leaf_spot": "Mulch heavily. Apply copper fungicide. Remove lower leaves.",
"Tomato___Spider_mites Two-spotted_spider_mite": "Spray water to dislodge. Apply insecticidal soap. Encourage predatory mites.",
"Tomato___Target_Spot": "Apply azoxystrobin. Improve air circulation. Remove infected leaves.",
"Tomato___Tomato_Yellow_Leaf_Curl_Virus": "Control whiteflies with yellow sticky traps. Use reflective mulch. Remove infected plants.",
"Tomato___Tomato_mosaic_virus": "Disinfect tools. Control aphids. Remove infected plants immediately."
}

def allowed_file(filename):
    return '.' in filename and \
           filename.rsplit('.', 1)[1].lower() in app.config['ALLOWED_EXTENSIONS']

def load_and_preprocess_image(image_path, target_size=(224, 224)):
    img = Image.open(image_path).convert('RGB')
    img = img.resize(target_size)
    img_array = np.array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array = img_array.astype('float32') / 255.
    return img_array

def draw_bounding_boxes(image_path, boxes, output_path):
    img = cv2.imread(image_path)
    for box in boxes:
        x_min, y_min, x_max, y_max = box
        cv2.rectangle(img, (x_min, y_min), (x_max, y_max), (0, 255, 0), 2)
        cv2.putText(img, 'Detected Area', (x_min, y_min - 10),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)
    cv2.imwrite(output_path, img)

def predict_image_class_and_boxes(image_path):
    img = load_and_preprocess_image(image_path)
    pred = model.predict(img)
    class_index = np.argmax(pred)
    label = classes[class_index]
    boxes = [[50, 50, 100, 100], [150, 150, 200, 200]]  # Example boxes
    return label, boxes

def send_sms(to_number, disease, suggestion):
    try:
        account_sid = os.getenv("TWILIO_SID")
        auth_token = os.getenv("TWILIO_AUTH_TOKEN")
        from_number = os.getenv("TWILIO_NUMBER")

        if not all([account_sid, auth_token, from_number]):
            print("❌ Missing Twilio credentials. Check environment variables.")
            return

        client = Client(account_sid, auth_token)
```

```
        message_body = f"🌿 CropProtect Report\nDisease: {disease}\nSuggestion:
{suggestion}"
        message = client.messages.create(
            body=message_body,
            from_=from_number,
            to=to_number
        )
        print(f"✅ SMS sent successfully. SID: {message.sid}")
    except Exception as e:
        print("❌ Error sending SMS:", str(e))

@app.route('/', methods=['GET', 'POST'])
def upload_file():
    if request.method == 'POST':
        if 'file' not in request.files:
            return redirect(request.url)

        file = request.files['file']
        if file.filename == '':
            return redirect(request.url)

        if file and allowed_file(file.filename):
            # Ensure upload directory exists
            os.makedirs(app.config['UPLOAD_FOLDER'], exist_ok=True)

            # Save original file
            filename = secure_filename(file.filename)
            filepath = os.path.join(app.config['UPLOAD_FOLDER'], filename)
            file.save(filepath)

            # Generate highlighted version
            highlighted_filename = f"highlighted_{filename}"
            highlighted_path = os.path.join(app.config['UPLOAD_FOLDER'],
highlighted_filename)

            # Predict and draw boxes
            label, boxes = predict_image_class_and_boxes(filepath)
            draw_bounding_boxes(filepath, boxes, highlighted_path)

            # Get suggestions and translations
            suggestion = suggestions.get(label, "No specific suggestion available.")
            language = request.form['language']
            phone = request.form['phone']

            readable_label = label.replace("___", " - ")
            translated_label = translator.translate(readable_label, dest=language).text
            translated_suggestion = translator.translate(suggestion, dest=language).text

            # Send SMS
            try:
                send_sms(phone, readable_label, suggestion)
            except Exception as e:
                print("❌ Failed to send SMS:", e)
```

```
            return render_template("app.html",
                                   label=readable_label,
                                   suggestion=suggestion,
                                   translated_label=translated_label,
                                   translated_suggestion=translated_suggestion,
                                   file_path=filename,
                                   highlighted_image_path=highlighted_filename)

    return render_template("app.html")

@app.route('/generate_report', methods=['POST'])
def generate_report():
    from reportlab.lib.styles import getSampleStyleSheet
    from reportlab.platypus import Paragraph
    from reportlab.lib.enums import TA_LEFT
    from reportlab.lib import colors
    from reportlab.lib.units import inch

    # Get form data
    label = request.form['label']
    suggestion = request.form['suggestion']
    translated_label = request.form.get('translated_label', '')
    translated_suggestion = request.form.get('translated_suggestion', '')
    file_path = request.form.get('file_path', '')

    buffer = BytesIO()
    p = canvas.Canvas(buffer, pagesize=A4)
    width, height = A4
    margin = 50
    y = height - margin

    # Title & Date
    p.setFont("Helvetica-Bold", 16)
    p.drawString(margin, y, " 🌿 CropVista - Disease Detection Report")
    y -= 25
    p.setFont("Helvetica", 10)
    p.drawString(margin, y, f"Date: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}")
    y -= 40

    # Function to wrap and draw text
    def draw_wrapped_text(canvas, title, text, x, y_start, width=480, font_size=11,
space_after=20):
        from reportlab.platypus import Paragraph
        from reportlab.lib.styles import getSampleStyleSheet
        from reportlab.platypus import Frame

        style = getSampleStyleSheet()["Normal"]
        style.fontSize = font_size
        style.leading = font_size + 2
        style.alignment = TA_LEFT

        para = Paragraph(f"<b>{title}</b><br/>{text}", style)
        f = Frame(x, y_start - 60, width, 60, showBoundary=0)
        f.addFromList([para], canvas)
        return y_start - 70
```

```
    # Text sections
    y = draw_wrapped_text(p, "🔍 Predicted Disease (English):", label, margin, y)
    y = draw_wrapped_text(p, "💊 Suggestion (English):", suggestion, margin, y)


    # Image
    if file_path:
        highlighted_path = os.path.join(app.config['UPLOAD_FOLDER'],
f"highlighted_{file_path}")
        if os.path.exists(highlighted_path):
            try:
                img = Image.open(highlighted_path)
                max_width = 4 * inch
                max_height = 4 * inch
                img.thumbnail((max_width, max_height))
                img_io = BytesIO()
                img.save(img_io, format='PNG')
                img_io.seek(0)
                y -= int(max_height) + 30
                p.drawImage(ImageReader(img_io), margin, y, width=max_width,
height=max_height)
            except Exception as e:
                print("❌ Error adding image to PDF:", e)

    p.showPage()
    p.save()
    buffer.seek(0)

    return send_file(buffer, as_attachment=True, download_name="CropProtect_Report.pdf",
mimetype='application/pdf')

if __name__ == '__main__':
    os.makedirs(app.config['UPLOAD_FOLDER'], exist_ok=True)
    app.run(host='0.0.0.0', port=3000, debug=True)
```

**app.html**

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>🌿 CropProtect - Crop Disease Detection</title>
  <link
href="https://fonts.googleapis.com/css2?family=Poppins:wght@300;500;700&display=swap"
rel="stylesheet" />
  <style>
    :root {
      --primary: #43a047;
      --dark-green: #2e7d32;
      --light-green: #e8f5e9;
      --blue: #1565c0;
      --hover-blue: #0d47a1;
      --bg-light: #f2f7f5;
```

```
  --bg-dark: #1e1e1e;
  --text-light: #000;
  --text-dark: #fff;
}

* {
  box-sizing: border-box;
  scroll-behavior: smooth;
}

body {
  font-family: 'Poppins', sans-serif;
  background: var(--bg-light);
  color: var(--text-light);
  margin: 0;
  transition: background 0.3s, color 0.3s;
}

body.dark-mode {
  background: var(--bg-dark);
  color: var(--text-dark);
}

/* NAVIGATION */
nav {
  position: sticky;
  top: 0;
  z-index: 999;
  background: white;
  display: flex;
  justify-content: space-between;
  align-items: center;
  padding: 12px 30px;
  box-shadow: 0 4px 12px rgba(0, 0, 0, 0.05);
  transition: background 0.4s;
}

body.dark-mode nav {
  background: #2c2c2c;
}

.nav-logo {
  font-weight: 700;
  font-size: 20px;
  color: var(--dark-green);
}

.nav-links {
  display: flex;
  gap: 25px;
  align-items: center;
}

.nav-links a {
  text-decoration: none;
```

```
  font-weight: 500;
  color: inherit;
  transition: color 0.2s;
  padding-bottom: 3px;
  position: relative;
}

.nav-links a::after {
  content: '';
  position: absolute;
  left: 0;
  bottom: 0;
  width: 0%;
  height: 2px;
  background: var(--dark-green);
  transition: width 0.3s ease;
}

.nav-links a:hover::after {
  width: 100%;
}

.dark-toggle-btn {
  background: none;
  border: none;
  font-size: 20px;
  cursor: pointer;
  margin-left: 10px;
  color: var(--blue);
  transition: transform 0.2s;
}

.dark-toggle-btn:hover {
  transform: scale(1.2);
}

/* MAIN CONTAINER */
.container {
  width: 95%;
  max-width: 720px;
  margin: 50px auto;
  background: white;
  padding: 40px;
  border-radius: 24px;
  box-shadow: 0 10px 25px rgba(0, 0, 0, 0.08);
  transition: background 0.3s;
}

body.dark-mode .container {
  background: #2c2c2c;
}

h1 {
  text-align: center;
  color: var(--dark-green);
```

```
  margin-bottom: 35px;
  font-weight: 700;
}

label {
  display: block;
  margin-top: 20px;
  font-weight: 500;
}

input, select, button {
  width: 100%;
  padding: 12px;
  margin-top: 6px;
  border-radius: 10px;
  border: 1px solid #ccc;
  font-size: 15px;
}

input[type="file"] {
  padding: 8px;
  background: #f9f9f9;
}

button {
  background-color: var(--primary);
  color: white;
  border: none;
  font-size: 16px;
  margin-top: 25px;
  cursor: pointer;
  transition: background-color 0.3s ease;
}

button:hover {
  background-color: #388e3c;
}

.image-preview-container {
  display: flex;
  flex-wrap: wrap;
  gap: 15px;
  margin-top: 15px;
}

.image-preview-container img {
  max-width: 48%;
  border-radius: 12px;
  box-shadow: 0 6px 15px rgba(0, 0, 0, 0.1);
}

.results {
  margin-top: 40px;
  padding: 25px;
  background: var(--light-green);
```

```
  border-left: 6px solid var(--dark-green);
  border-radius: 14px;
  animation: fadeIn 0.6s ease-in-out;
}

.results h3 {
  margin: 14px 0 6px;
  color: var(--dark-green);
}

.results img {
  margin-top: 20px;
  max-width: 100%;
  border-radius: 16px;
  box-shadow: 0 10px 20px rgba(0, 0, 0, 0.1);
}

.report-btn {
  background-color: var(--blue);
  margin-top: 20px;
}

.report-btn:hover {
  background-color: var(--hover-blue);
}

.spinner {
  margin: 20px auto;
  width: 40px;
  height: 40px;
  border: 4px solid #c8e6c9;
  border-top: 4px solid var(--dark-green);
  border-radius: 50%;
  animation: spin 1s linear infinite;
}

@keyframes spin {
  to { transform: rotate(360deg); }
}

@keyframes fadeIn {
  from { opacity: 0; transform: translateY(20px); }
  to { opacity: 1; transform: translateY(0); }
}

@media (max-width: 650px) {
  nav {
    flex-direction: column;
    gap: 8px;
    padding: 12px;
  }

  .nav-links {
    flex-wrap: wrap;
    justify-content: center;
```

```
    }

    .image-preview-container img {
      max-width: 100%;
    }
  }
 </style>
</head>
<body>
 <nav>
  <div class="nav-logo">🌾 CropProtect</div>
  <div class="nav-links">
   <a href="#home">Home</a>
   <a href="#about">About</a>
   <a href="#contact">Contact</a>
   <a href="#predict">Predict</a>
   <button class="dark-toggle-btn" onclick="toggleDarkMode()">🌙</button>
  </div>
 </nav>

 <div class="container" id="home">
  <h1>🌾 CropProtect - Disease Predictor</h1>
  <form action="/" method="POST" enctype="multipart/form-data"
onsubmit="showSpinner()" id="predict">
   <label>📤 Upload Plant Images:</label>
   <input type="file" name="file" accept="image/*" required
onchange="previewImages(event)">
   <div class="image-preview-container" id="imagePreviewContainer"></div>

   <label>📱 Your Phone Number:</label>
   <input type="text" name="phone" placeholder="+91XXXXXXXXXX" required>

   <label>🌐 Preferred Language:</label>
   <select name="language" required>
    <option value="en">GB English</option>
    <option value="hi">IN Hindi</option>
    <option value="te">IN Telugu</option>
    <option value="ta">IN Tamil</option>
    <option value="kn">IN Kannada</option>
   </select>

   <button type="submit">🔍 Predict & Notify</button>
   <div id="spinner" class="spinner" style="display: none;"></div>
  </form>

  {% if label %}
  <div class="results">
   <h3>🔍 Predicted Disease (English):</h3>
   <p>{{ label }}</p>

   <h3>💊 Suggestion (English):</h3>
   <p>{{ suggestion }}</p>
```

```
    <h3>🌐 Translated Disease:</h3>
    <p>{{ translated_label }}</p>

    <h3>🌱 Translated Suggestion:</h3>
    <p>{{ translated_suggestion }}</p>

    {% if highlighted_image_path %}
    <img src="{{ url_for('static', filename=highlighted_image_path) }}" alt="Highlighted
Image">
    {% endif %}

    <form action="/generate_report" method="POST">
      <input type="hidden" name="label" value="{{ label }}">
      <input type="hidden" name="suggestion" value="{{ suggestion }}">
      <input type="hidden" name="translated_label" value="{{ translated_label }}">
      <input type="hidden" name="translated_suggestion" value="{{ translated_suggestion
}}">
      <input type="hidden" name="file_path" value="{{ file_path }}">
      <button class="report-btn" type="submit">📄 Generate Report</button>
    </form>
   </div>
   {% endif %}
  </div>

  <div class="container" id="about">
   <h2>📘 About</h2>
   <p>CropProtect uses AI to detect plant diseases from leaf images and provides helpful
treatment suggestions in your local language.</p>
  </div>

  <div class="container" id="contact">
   <h2>📞 Contact Us</h2>
   <p>Email: support@CropProtect.ai</p>
   <p>Phone: +91 98765 43210</p>
  </div>

  <script>
   function toggleDarkMode() {
    const body = document.body;
    const btn = document.querySelector('.dark-toggle-btn');
    body.classList.toggle("dark-mode");
    btn.textContent = body.classList.contains("dark-mode") ? "🌞" : "🌙";
   }

   function previewImages(event) {
    const container = document.getElementById('imagePreviewContainer');
    container.innerHTML = '';
    const file = event.target.files[0];
    if (file) {
     const reader = new FileReader();
     reader.onload = function(e) {
      const img = document.createElement('img');
      img.src = e.target.result;
```

```
        container.appendChild(img);
      };
      reader.readAsDataURL(file);
    }
  }

  function showSpinner() {
    document.getElementById('spinner').style.display = 'block';
  }
 </script>
</body>
</html>
```

Cropdisease.ipynb:

```
import os
!ls PlantVillage-Dataset
os.listdir("PlantVillage-Dataset")
!ls PlantVillage-Dataset/raw/color
source_dir = 'PlantVillage-Dataset/raw/color'
import shutil
import os

source_dir = 'PlantVillage-Dataset/raw/color'
target_dir = '/content/dataset/'

os.makedirs(target_dir, exist_ok=True)

# List of folders
selected_classes = [
  'Apple___Apple_scab',
  'Apple___Black_rot',
  'Apple___Cedar_apple_rust',
  'Apple___healthy',
  'Blueberry___healthy',
  'Cherry_(including_sour)___healthy',
  'Cherry_(including_sour)___Powdery_mildew',
  'Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot',
  'Corn_(maize)___Common_rust_',
  'Corn_(maize)___healthy',
  'Corn_(maize)___Northern_Leaf_Blight',
  'Grape___Black_rot',
  'Grape___Esca_(Black_Measles)',
  'Grape___healthy',
  'Grape___Leaf_blight_(Isariopsis_Leaf_Spot)',
  'Orange___Haunglongbing_(Citrus_greening)',
  'Peach___Bacterial_spot',
  'Peach___healthy',
  'Pepper,_bell___Bacterial_spot',
  'Pepper,_bell___healthy',
  'Potato___Early_blight',
  'Potato___healthy',
  'Potato___Late_blight',
  'Raspberry___healthy',
  'Soybean___healthy',
```

```
    'Squash___Powdery_mildew',
    'Strawberry___healthy',
    'Strawberry___Leaf_scorch',
    'Tomato___Bacterial_spot',
    'Tomato___Early_blight',
    'Tomato___healthy',
    'Tomato___Late_blight',
    'Tomato___Leaf_Mold',
    'Tomato___Septoria_leaf_spot',
    'Tomato___Spider_mites Two-spotted_spider_mite',
    'Tomato___Target_Spot',
    'Tomato___Tomato_mosaic_virus',
    'Tomato___Tomato_Yellow_Leaf_Curl_Virus',
]
for cls in selected_classes:
    src_path = os.path.join(source_dir, cls)
    dst_path = os.path.join(target_dir, cls)
    if os.path.exists(src_path):
        shutil.copytree(src_path, dst_path)
    else:
        print(f"Not found: {cls}")

print("Copy complete.")
import os
folder_path = 'PlantVillage-Dataset/raw/color'
classes = os.listdir(folder_path)
print("Available class folders:")
for cls in classes:
    print(cls)
from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(
    rescale=1./255,
    validation_split=0.2,
    rotation_range=15,
    zoom_range=0.1,
    horizontal_flip=True
)

train_gen = train_datagen.flow_from_directory(
    '/content/dataset',
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical',
    subset='training'
)

val_gen = train_datagen.flow_from_directory(
    '/content/dataset',
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical',
    subset='validation'
)
from tensorflow.keras.applications import MobileNetV2
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.optimizers import Adam

# Load MobileNetV2 without the top layer
base_model = MobileNetV2(input_shape=(224, 224, 3), include_top=False,
weights='imagenet')
base_model.trainable = False  # Freeze base model layers

# Build model
model = Sequential([
   base_model,
   GlobalAveragePooling2D(),
   Dense(128, activation='relu'),
   Dense(train_gen.num_classes, activation='softmax')
])

# Compile
model.compile(optimizer=Adam(),
            loss='categorical_crossentropy',
            metrics=['accuracy'])

# Summary
model.summary()

EPOCHS = 11
history = model.fit(
   train_gen,
   validation_data=val_gen,
   epochs=EPOCHS
)

 import matplotlib.pyplot as plt

 # Plot accuracy
 plt.plot(history.history['accuracy'], label='Training Accuracy')
 plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
 plt.title("Model Accuracy")
 plt.xlabel("Epochs")
 plt.ylabel("Accuracy")
 plt.legend()
 plt.grid(True)
 plt.show()

model.save("crop_disease.h5")
print("Model saved successfully!")
```

## 4.2 IMPLEMENTATION

Create a directory folder as following and copy relevant pieces of code into the required parts: -

### 4.2.1 Directory folder

CROP-DISEASE-PREDICITON/

```
├── app.py
│
├── static/
│   ├── css/
│   └── style.css
│
├── templates/
│   ├── app.html
├── crop_disease.h5
```

### 4.2.2 Model Development on Google Colab

The model was developed and trained on **Google Colab**, a cloud-based Python environment ideal for deep learning applications. The following steps outline the model creation and training process:

1. **Dataset Loading and Preprocessing:**

   A large dataset of crop leaf images with multiple disease classes was loaded. The dataset was split into training and validation sets. Images were resized to a standard shape (224x224) and normalized to ensure compatibility with the CNN input layer.

2. **Model Architecture MobileNetV2:**

   A transfer learning approach was adopted using the MobileNetV2 architecture as a base model. The final dense layers were customized to match the number of disease categories.

3. **Training Strategy**
   - Optimizer: Adam
   - Loss Function: Categorical Crossentropy
   - Metrics:

     Accuracy

     The model was trained for multiple epochs until convergence was achieved with minimal validation loss.

4. **Evaluation and Export**

   The model was evaluated using unseen validation data, and achieved high accuracy. It was then saved as plant_disease_prediction.h5 using:

python

model.save("crop_disease.h5")

### 4.2.3 Integration and Deployment

Once trained, the model was integrated into the Flask application via app.py. The web interface allows farmers to upload leaf images, which are then processed by the model to predict the disease. The following additional integrations were made:

- **Multilingual Translation**: Google Translate API was used to translate disease descriptions and treatment instructions into the farmer's local language based on user input.
- **SMS Notification**: Twilio API was configured to send the disease name and translated treatment details as an SMS to the farmer's registered phone number.

### 4.2.4 User Interaction Flow

1. The farmer visits the web app and uploads an image of a diseased leaf.
2. The image is processed and fed into the trained CNN model.
3. The model returns the predicted disease name.
4. A description and treatment recommendation are fetched and translated to the user's preferred language.
5. A multilingual report is displayed on the web and sent to the farmer's mobile phone via SMS.

# 5. TESTING

## 5.1 INTRODUCTION TO TESTING

Testing is a critical aspect of software development aimed at verifying, validating, and ensuring the quality and reliability of various software components. It helps to minimize risks and optimize resource utilization throughout the development lifecycle. Although testing can be applied at any stage, implementing it early allows for the identification and resolution of defects before they escalate. Testing involves evaluating the software under different conditions and environments to assess its functionality, performance, and other essential attributes. Various testing methodologies are employed depending on the nature and objectives of the software being developed. In this project, the testing phase ensures that every module functions effectively, thereby contributing to the system's overall reliability and performance.

In this project, testing was conducted to verify the end-to-end performance of the **AI-powered plant disease detection and prevention system**, including its ability to:

- Correctly predict diseases from crop leaf images,
- Translate the diagnosis and recommendation into the preferred user language,
- Deliver results seamlessly via the web interface and SMS notification module.

The testing phase focused not only on verifying the accuracy of the deep learning model's predictions but also on ensuring the smooth integration of auxiliary modules such as image preprocessing, translation, and notification systems.

The primary goals of testing in this project were to:

- Ensure high prediction accuracy for multiple crop disease classes.
- Validate the correctness and completeness of the translated report.
- Verify the responsiveness and usability of the web interface under different input conditions.
- Confirm successful SMS delivery through Twilio for real-time farmer notification.
- Assess the overall robustness and fault tolerance of the system.

Each test case was executed systematically, and the results were thoroughly documented to compare actual outcomes against expected results.

## 5.2 TEST CASES

Table 5.1 Test Cases

| Test Case ID | Test Case Name | Test Description | Expected Output | Actual Output | Remark |
|---|---|---|---|---|---|
| TC_01 | Healthy Leaf Detection | Upload a healthy leaf image (e.g., Tomato_healthy.jpg) | Label displayed: "Tomato – healthy" | Leaf shown as healthy | Success |
| TC_02 | Unhealthy Leaf Detection | Upload a diseased leaf image (e.g., Potato_Early_blight.jpg) | Disease name detected and displayed | Disease shown on dashboard | Success |
| TC_03 | SMS Notification | Enter user phone number after detection | SMS delivered with disease name and suggestions | SMS delivered successfully | Success |
| TC_04 | PDF Report Generation | Click "Generate Report" after detection | PDF generated with image and disease information | PDF generated correctly | Success |
| TC_05 | Multilingual Translation | Select any target language (e.g., Hindi) for output | Information translated into selected language | Report shown in that language | Success |

Table 5.1, shows the results of the test cases completed. It can be concluded that the system redirects properly and there are no issues in predicting and sending the notification.

# 6. RESULTS

## Performance Analysis

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Apple___Apple_scab | 1.00 | 0.98 | 0.99 | 63 |
| Apple___Black_rot | 1.00 | 1.00 | 1.00 | 62 |
| Apple___Cedar_apple_rust | 1.00 | 1.00 | 1.00 | 27 |
| Apple___healthy | 0.99 | 1.00 | 1.00 | 165 |
| Blueberry___healthy | 1.00 | 1.00 | 1.00 | 158 |
| Cherry_(including_sour)___Powdery_mildew | 1.00 | 1.00 | 1.00 | 105 |
| Cherry_(including_sour)___healthy | 1.00 | 1.00 | 1.00 | 85 |
| Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot | 0.98 | 0.88 | 0.93 | 51 |
| Corn_(maize)___Common_rust_ | 1.00 | 1.00 | 1.00 | 119 |
| Corn_(maize)___Northern_Leaf_Blight | 0.94 | 0.99 | 0.97 | 98 |
| Corn_(maize)___healthy | 1.00 | 1.00 | 1.00 | 116 |
| Grape___Black_rot | 1.00 | 0.98 | 0.99 | 118 |
| Grape___Esca_(Black_Measles) | 0.99 | 1.00 | 0.99 | 139 |
| Grape___Leaf_blight_(Isariopsis_Leaf_Spot) | 1.00 | 1.00 | 1.00 | 108 |
| Grape___healthy | 1.00 | 1.00 | 1.00 | 42 |
| Orange___Haunglongbing_(Citrus_greening) | 1.00 | 1.00 | 1.00 | 551 |
| Peach___Bacterial_spot | 1.00 | 1.00 | 1.00 | 238 |
| Peach___healthy | 1.00 | 1.00 | 1.00 | 36 |
| Pepper,_bell___Bacterial_spot | 1.00 | 1.00 | 1.00 | 108 |
| Pepper,_bell___healthy | 1.00 | 0.99 | 1.00 | 148 |
| Potato___Early_blight | 1.00 | 1.00 | 1.00 | 108 |
| Potato___Late_blight | 1.00 | 1.00 | 1.00 | 108 |
| Potato___healthy | 1.00 | 1.00 | 1.00 | 15 |
| Raspberry___healthy | 1.00 | 1.00 | 1.00 | 37 |
| Soybean___healthy | 1.00 | 1.00 | 1.00 | 509 |
| Squash___Powdery_mildew | 1.00 | 1.00 | 1.00 | 184 |
| Strawberry___Leaf_scorch | 1.00 | 1.00 | 1.00 | 111 |
| Strawberry___healthy | 1.00 | 1.00 | 1.00 | 45 |
| Tomato___Bacterial_spot | 1.00 | 1.00 | 1.00 | 213 |
| Tomato___Early_blight | 1.00 | 0.99 | 0.99 | 108 |
| Tomato___Late_blight | 0.98 | 0.99 | 0.99 | 191 |
| Tomato___Leaf_Mold | 1.00 | 0.99 | 0.99 | 95 |
| Tomato___Septoria_leaf_spot | 0.99 | 0.99 | 0.99 | 177 |
| Tomato___Spider_mites Two-spotted_spider_mite | 1.00 | 0.98 | 0.99 | 168 |
| Tomato___Target_Spot | 0.98 | 1.00 | 0.99 | 141 |
| Tomato___Tomato_Yellow_Leaf_Curl_Virus | 1.00 | 1.00 | 1.00 | 536 |
| Tomato___Tomato_mosaic_virus | 1.00 | 1.00 | 1.00 | 37 |
| Tomato___healthy | 0.99 | 1.00 | 1.00 | 159 |
| accuracy | | | 1.00 | 5431 |
| macro avg | 1.00 | 0.99 | 0.99 | 5431 |
| weighted avg | 1.00 | 1.00 | 1.00 | 5431 |

Fig 6.1: Performance metrics like precision, recall, f1-score.

Fig 6.1 shows the performance metrics of the project like precision, accuracy , f1-score and recall. From the figure, we can identify how the model is performing.

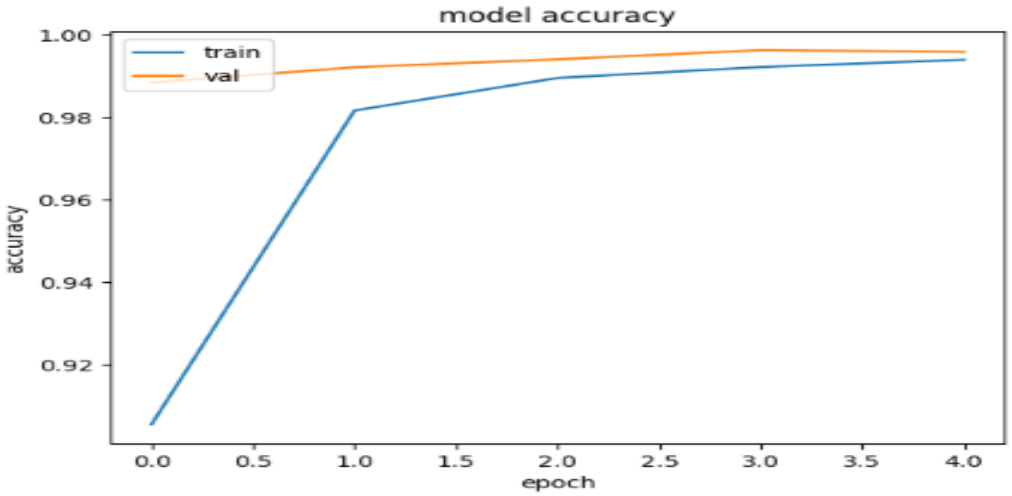



Fig 6.2 Model accuracy during training and validation phase

Fig 6.2 shows the graph between the accuracy during training and validation phase. This figure shows how the model behave during training and validation phase.

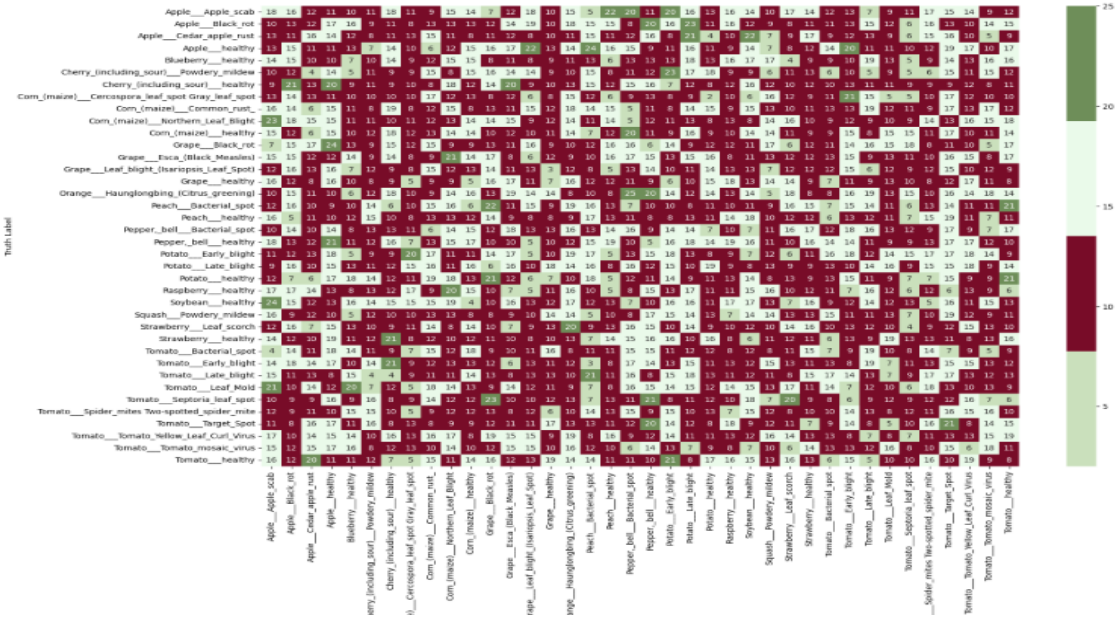


Fig 6.3 Confusion matrix of Crop Disease Prediction System

Fig 6.3 shows the confusion matrix of the project. It has plant diseases as rows and columns.

## Website Results

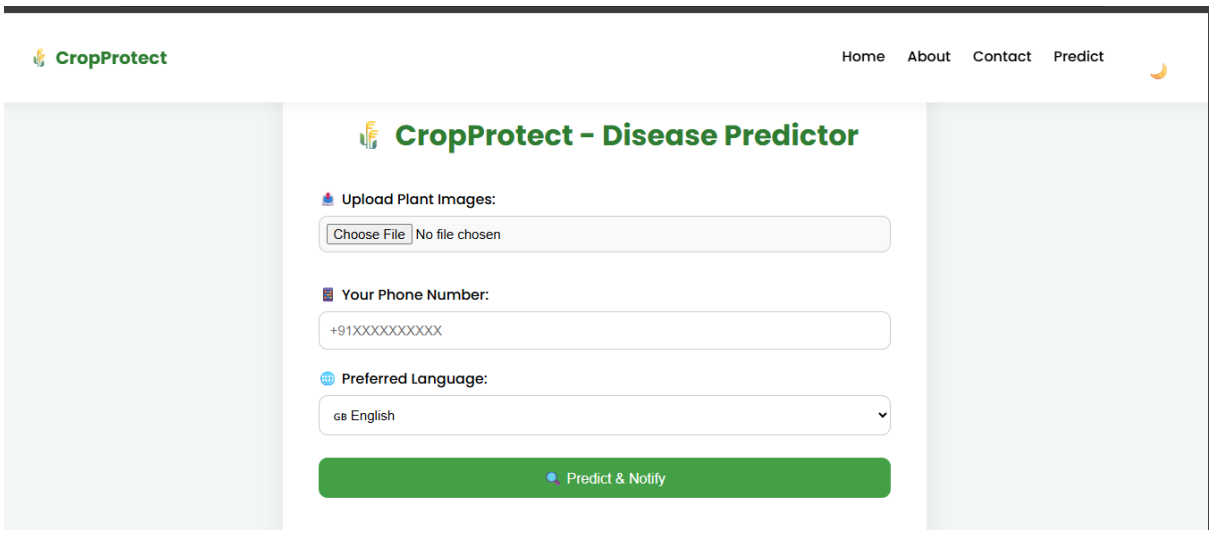



Fig. 6.4 Home page of the websitez

Fig. 6.4 shows the initial page displayed upon initially loading the local host address. In the home page, the user has to upload the crop image along with the mobile number and can select the preferred language.

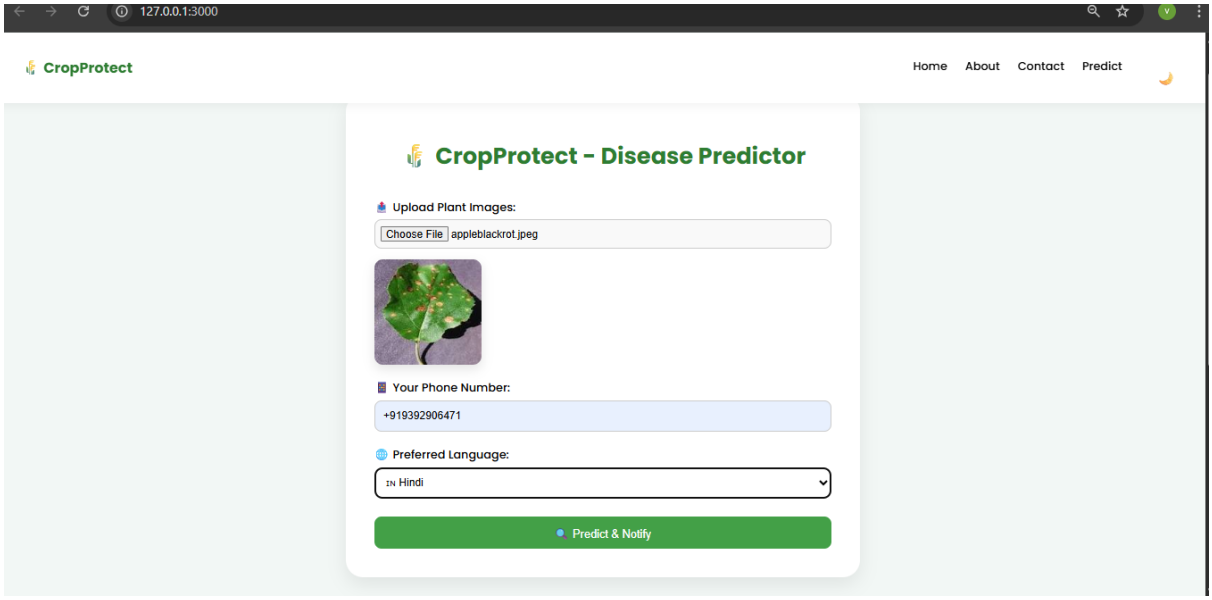



Fig. 6.5 User uploading the crop image

Fig. 6.5 shows the user has uploaded a unhealthy leaf image and the user has given phone number on which the user will get the notification.

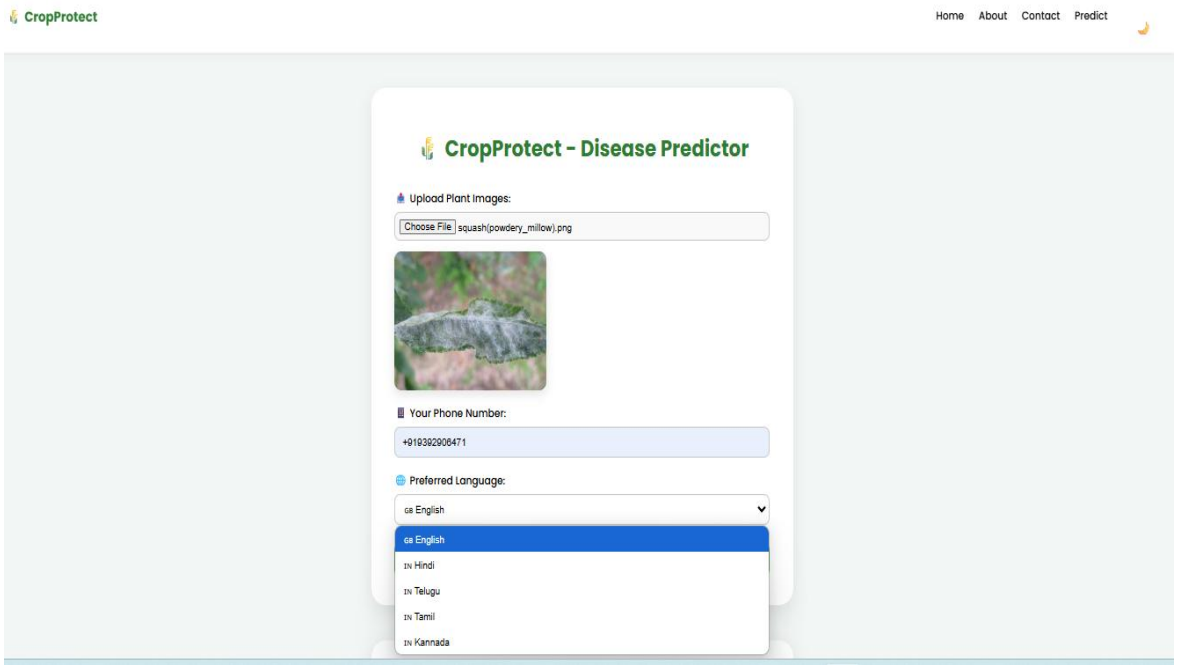



Fig 6.6 Drop-down menu for languages

Fig 6.6 shows a drop-down menu which provides the user options of 5 languages.

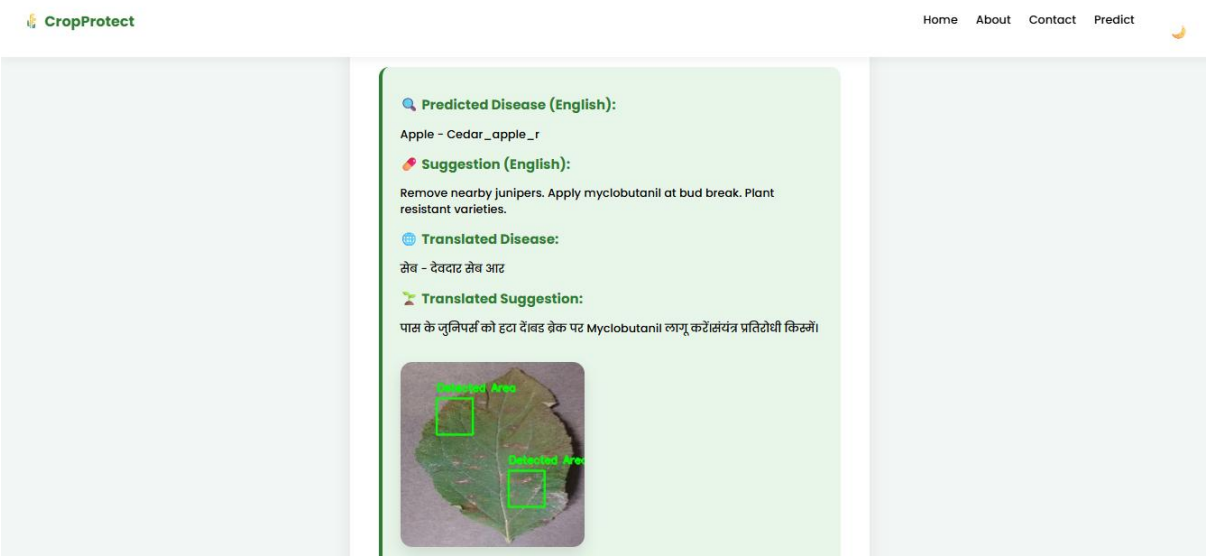



Fig. 6.7 Result Page

Fig. 6.7 shows the result of the leaf image given by the user along with the language he has chosen and the user can get the report by clicking on the “generate report”.


Fig. 6.8 Notification received on user’s mobile device

Fig. 6.4 shows the notification received by the user on his registered mobile device

# 7. CONCLUSION AND FUTURE ENHANCEMENTS

## 7.1 CONCLUSION

The CropProtect system effectively demonstrates the application of AI-driven techniques to address critical challenges in plant disease management within agriculture. By integrating a convolutional neural network for image-based disease detection with support modules such as SMS notifications, multilingual output, and automated PDF reporting, the system delivers a comprehensive and user-centric solution. These features not only enable timely and accurate disease diagnosis but also ensure that the information is accessible and actionable for farmers across different linguistic and technological backgrounds.

The design choices made in this project—particularly the inclusion of end-to-end automation and field communication—reflect a focus on practical deployment and scalability. The multilingual and SMS support enhances reach among rural and local farming communities, while the AI model ensures reliable performance in diverse real-world conditions. As a result, CropProtect not only bridges the gap between modern AI technologies and traditional farming practices but also lays the groundwork for future developments in precision agriculture and farmer-centered innovation.

## 7.2 FUTURE ENHANCEMENT

To further extend the capabilities of the CropProtect system, several innovative enhancements are envisioned. Integrating IoT-based monitoring using sensors for temperature, humidity, and soil moisture will enable continuous real-time environmental tracking, providing vital context for more accurate disease prediction. A dedicated mobile application equipped with camera support can empower farmers to diagnose plant health issues directly from the field, enhancing accessibility and responsiveness. Incorporating GPS and weather data will allow the system to deliver localized crop care tips, improving regional adaptability. Moreover, the addition of a voice assistant that supports regional languages can bridge digital literacy gaps, making the platform inclusive for non-tech-savvy users. Introducing a farmer feedback module will enable iterative improvements and better trust in the system's predictions. Future versions may also cover a wider range of crops and rare plant diseases, ensuring broader agricultural impact.

# REFERENCES


[1] Seyed Mohamad Javidan, Ahmad Banakar, Kamran Rahnama, Keyvan Asefpour Vakilian, Yiannis Ampatzidis,Feature engineering to identify plant diseases using image processing and artificial intelligence: A comprehensive review, Smart Agricultural Technology, Volume8, 2024, 100480,ISSN 2772-3755, https://doi.org/10.1016/j.atech.2024.100480.

[2] Divyanshu Tirkey, Kshitiz Kumar Singh, Shrivishal Tripathi, Performance analysis of AI-based solutions for cropdisease identification, detection, and classification, Smart Agricultural Technology, Volume 5, 2023, 100238, ISSN 2772-3755, https://doi.org/10.1016/j.atech.2023.100238.

[3] Sushruta Mishra, Dayal Rohan Volety, Navdeep Bohra, Sultan Alfarhood, Mejdl Safran, A smart and sustainable framework for millet crop monitoring equipped with disease detection using enhanced predictive intelligence, Alexandria Engineering Journal, Volume 83, 2023, Pages 298-306, ISSN 1110-0168, https://doi.org/10.1016/j.aej.2023.10.041.

[4] V. Balafas, E. Karantoumanis, M. Louta and N. Ploskas, "Machine Learning and Deep Learning for Plant Disease Classification and Detection," in IEEE Access, vol. 11, pp. 114352-114377, 2023, doi: 10.1109/ACCESS.2023.3324722.

[5] Dolatabadian A, Neik TX, Danilevicz MF, Upadhyaya SR, Batley J, Edwards D. Image-based crop disease detection using machine learning. Plant Pathology. 2025 Jan;74(1):18-38

[6] Plantix by PEAT GmbH. A smartphone-based app using AI for plant disease, pest, and nutrient deficiency detection. Available: https://plantix.net/en/ (Accessed April 2025).

[7] AgroDoctor. A database-driven plant disease diagnosis and treatment platform. Not AI-based or real-time. Available: https://www.agrodoctor.eu/ (Accessed April 2025).

[8] CropIn - SmartFarm Platform. AI-powered farm management and disease alert system integrating IoT and satellite imagery. Available: https://www.cropin.com/solutions/smartfarm (Accessed April 2025).

[9] Khamparia, A., Saini, G., Gupta, D., Khanna, A., Tiwari, S. and de Albuquerque, V.H.C., 2020. Seasonal crops disease prediction and classification using deep convolutional encoder network. *Circuits, Systems, and Signal Processing*, *39*, pp.818-836.

[10] S. S. Shinde and M. Kulkarni, "Review Paper on Prediction of Crop Disease Using IoT and Machine Learning," 2017 International Conference on Transforming Engineering Education (ICTEE), Pune, India, 2017, pp. 1-4, doi: 10.1109/ICTEED.2017.8586207. keywords: {Hafnium;Agriculture;Internet of Things;Machine Learning;Prediction algorithms;Sensors},

[11] Pokkuluri, K.S., Nedunuri, S.U.D. and Devi, U., 2022. Crop Disease Prediction with Convolution Neural Network (CNN) Augmented With Cellular Automata. *Int. Arab J. Inf. Technol.*, *19*(5), pp.765-773.

[12] Domingues, T., Brandão, T. and Ferreira, J.C., 2022. Machine learning for detection and prediction of crop diseases and pests: A comprehensive survey. *Agriculture*, *12*(9), p.1350.

[13] Ashraf, M., Abrar, M., Qadeer, N., Alshdadi, A.A., Sabbah, T. and Khan, M.A., 2023. A Convolutional Neural Network Model for Wheat Crop Disease Prediction. *Computers, Materials & Continua*, *75*(2).

[14] Nandhini, S. and Ashokkumar, K., 2022. Machine learning technique for crop disease prediction through crop leaf image. *Appl. Math. Inf. Sci*, *16*(2), pp.149-158.

[15] Jain, S. and Ramesh, D., 2021, July. AI based hybrid CNN-LSTM model for crop disease prediction: An ML advent for rice crop. In *2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT)* (pp. 1-7). IEEE.

[16] Fenu, G. and Malloci, F.M., 2021. Forecasting plant and crop disease: an explorative study on current algorithms. *Big Data and Cognitive Computing*, *5*(1), p.2.