# PLANT DISEASE DETECTION USING IMAGES

# MINI PROJECT REPORT

***Submitted by***

# R M VIJAYBALAJI 211422104921

**V.HARISH** **211422104911**

***in partial fulfillment for the award of the degree of***

# BACHELOR OF ENGINEERING

***in***

# COMPUTER SCIENCE AND ENGINEERING



**PANIMALAR ENGINEERING COLLEGE**

**(An Autonomous Institution, Affiliated to Anna University, Chennai)**

**OCTOBER 2025**

# BONAFIDE CERTIFICATE

Certified that this project report “**PLANT DISEASE DETECTION USING IMAGES”** is the bonafide work of **R.M.VIJAYBALAJI(211422104921)** & **V.HARISH(211422104911)** who carried out the project work under my supervision.

| **SIGNATURE**  **Dr.L.JABASHEELA,M.E.,Ph.D., PROFESSOR,**  **HEAD OF THE DEPARTMENT**  DEPARTMENT OF CSE,  PANIMALAR ENGINEERING COLLEGE,  NASARATHPETTAI,POONAMALLEE, CHENNAI-600 123. | **SIGNATURE**  **Mr.P.PRABBU SANKAR, M.E.,(Ph.D.,) ASSISTANT PROFESSOR, SUPERVISOR**  DEPARTMENT OF CSE,  PANIMALAR ENGINEERING COLLEGE,  NASARATHPETTAI,POONAMALLEE, CHENNAI-600 123. |
| --- | --- |

Certified that the above candidate(s) were examined in the Mini Project Viva-Voce Examination held on...........................

**INTERNAL EXAMINER EXTERNAL EXAMINER**

# ACKNOWLEDGEMENT

We would like to express our deep gratitude to our respected Secretary and Correspondent **Dr.P.CHINNADURAI, M.A., Ph.D.** for his kind words and enthusiastic motivation, which inspired us a lot in completing this project.

We express our sincere thanks to our beloved Directors Tmt. **C.VIJAYARAJESWARI, Dr.C.SAKTHI KUMAR,M.E.,Ph.D** and **Dr.SARANYASRESSSAKTHI**

**KUMAR.,B.E,M.B.A Ph.D.,** for providing us with the necessary facilities to undertake this project.

We also express our gratitude to our Principal **Dr.K.MANI, M.E., Ph.D.** who facilitated us in completing the project.

We thank the Head of the CSE Department, **Dr. L.JABASHEELA, M.E.,Ph.D.,** for the support extended throughout the project.

We would like to thank our parents, friends, project Guide **Mr.PRABBU SANKAR,M.E.,(Ph.D.,)**and coordinator **Dr.A.HEMLATHADHEVI,M.E.,Ph.D.,**

**PROFESSOR ,** and all the faculty members of the Department of CSE for their advice and encouragement for the successful completion of the project.

**SIGNATURE OF THE STUDENTS**

**R.M.VIJAYBALAJI(211423104921)**

**V.HARISH(211423104911)**

# ABSTRACT

Crop diseases are a noteworthy risk to sustenance security, however their quick distinguishing proof stays troublesome in numerous parts of the world because of the non attendance of the important foundation. Emergence of accurate techniques in the field of leaf-based image classification has shown impressive results. This paper makes use of Random Forest in identifying between healthy and diseased leaf from the data sets created. Our proposed paper includes various phases of implementation namely dataset creation, feature extraction, training the classifier and classification. The created datasets of diseased and healthy leaves are collectively trained under Random Forest to classify the diseased and healthy images. For extracting features of an image we use Histogram of an Oriented Gradient (HOG). Overall, using machine learning to train the large data sets available publicly gives us a clear way to detect the disease present in plants in a colossal scale.

**TABLE CONTENTS**

| **CHAPTER NO** | **TITLE** | **PAGE NO** |
| --- | --- | --- |
|  | **ABSTRACT** | ix |
|  | **LIST OF TABLES** | vii |
|  | **LIST OF FIGURES** | viii |
| **1.** | **INTRODUCTION** | 01 |
|  | 1.1 Overview | 01 |
|  | 1.2 Problem Definition | 02 |
| **2.** | **SYSTEM ANALYSIS** | 03 |
|  | 2.1 Existing System | 03 |
|  | 2.2 Proposed System | 03 |
|  | 2.3 Development Environment | 04 |
| **3.** | **SYSTEM DESIGN** | 05 |
|  | 3.1 UML Diagrams | 05 |
|  | 3.2 Visul Studio Code | 09 |
|  | 3.4 Data Flow Diagram | 11 |
| **4.** | **SYSTEM ARCHITECTURE** | 13 |
|  | 4.1 Architecture Overview | 17 |
|  | 4.2 Module Description | 14 |
| **5.** | **SYSTEM IMPLEMENTATION** | 21 |
|  | 5.1 Implementation | 16 |

|  | 5.2 Test Data And Output | 17 |
| --- | --- | --- |
|  | 5.3 Coding | 23 |
| **6.** | **SYSTEM TESTING** | 37 |
|  | 6.1 Test Cases and Reports | 30 |
| **7.** | **CONCLUSION** | 36 |
|  | 7.1 Conclusion | 36 |
|  | 7.2 Future enhancement | 36 |
| **8.** | **APPENDICES** | 38 |
|  | Sample Screenshots | 32 |
| **9.** | **REFERENCES** | 44 |

**LIST OF FIGURES**

| **FIG NO** | **FIGURE DESCRIPTION** | **PAGE NO** |
| --- | --- | --- |
| 3.1.1 | Use Case Diagram for Plant disease detection | 05 |
| 3.1.2 | Class Diagram for Plant disease detection | 06 |
| 3.1.3 | Sequence Diagram for Plant disease detection | 07 |
| 3.1.4 | Collaboration Diagram for Plant disease detection | 08 |
| 3.1.5 | Activity Diagram for Plant disease detection | 10 |
| 3.3 | Dataflow Diagram | 13 |
| 3.4.1 | Architecture Diagram | 14 |

**CHAPTER 1**  **INTRODUCTION**

## OVERVIEW

In India about 70% of the populace relies on agriculture. Identification of the plant diseases is important in order to prevent the losses within the yield. It’s terribly troublesome to observe the plant diseases manually. It needs tremendous quantity of labor, expertize within the plant diseases, and conjointly need the excessive time interval. Hence, image processing and machine learning models can be employed for the detection of plant diseases. In this project, we have described the technique for the detection of plant diseases with the help of their leaves pictures. Image processing is a branch of signal processing which can extract the image properties or useful information from the image. Machine learning is a sub part of artificial intelligence which works automatically or give instructions to do a particular task. The main aim of machine learning is to understand the training data and fit that training data into models that should be useful to the people.

* 1. **PROBLEM DEFINITION**

The problem definition for using images to detect plant diseases is to develop an automated and accurate system that can identify, classify, and potentially quantify diseases by analyzing visual patterns in plant images. This technology aims to provide a faster, more cost-effective, and scalable alternative to traditional, time-consuming manual inspection by human experts.The core problem is that manual plant disease detection is inefficient and prone to human error, particularly on large farms, which leads to significant crop yield loss and negative economic impact. An automated system is needed to help farmers, especially in resource-limited areas, manage crop health proactively and sustainably.

**CHAPTER 2**

**SYSTEM ANALYSIS**

## EXISTING SYSTEM

Now a day’s, dilettante farmers are hard to understand the cultivation process, crop, climate change, etc. Farming is that the spine for every nation's economy. Future agriculture depends on dilettante formers. But, the new farmers have low level knowledge in this field. So, Machine learning help to solve this type of problems. In Existing system they provide soil type and crop using Random Forest algorithm. But everyone can able to find a soil type easily. So, we need to predict the crop type and predict the crop price based on Machine learning technology.

## PROPOSED SYSTEM

The system prepared predict major crops yield in a particular district in Tamil Nadu. The client on their first login has to register themselves on the application on android phone .Once the user logins into the system he gets all the access for predicting crop yield and using the input such as location, temperature, pH value, rainfall and humidity depends on their forming land environment. After submitting the inputs, it’s redirect into Firebase. The firebase is an intermediate between user input and trained data set. The input goes to the trained data, where it processes random forest algorithm to predict crop and price. After the prediction, the predicted value passes to the fire base. That firebase gives the predict value to the user on android application.

**ADVANTAGES**

* + 1. Improved accuracy and consistency
    2. Early and timely detection
  1. **DEVELOPMENT ENVIRONMENT**

## SOFTWARE REQUIREMENT

* Front End – HTML, CSS, JavaScript
* Backend – Node.js
* Language – Python 3.12

## HARDWARE REQUIREMENT

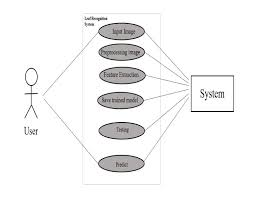
* Hard Disk: 100 GB SSD or more
* RAM: Minimum 8–16 GB
* Processor: Dual-core or higher

**CHAPTER 3**

**SYSTEM DESIGN**

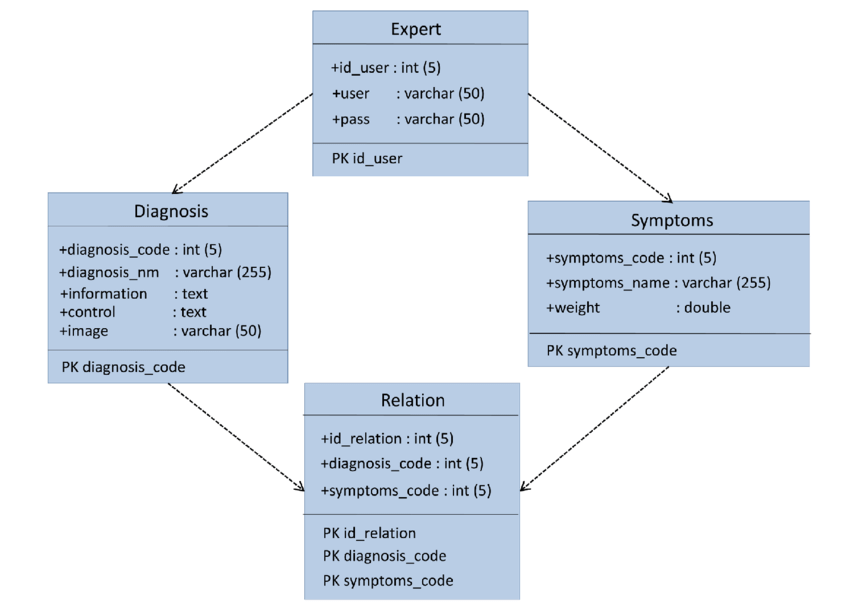
# UML DIAGRAMS

**Use Case Diagram**

****

**Fig 3.1.1 Use case diagram for Plant disease detection**

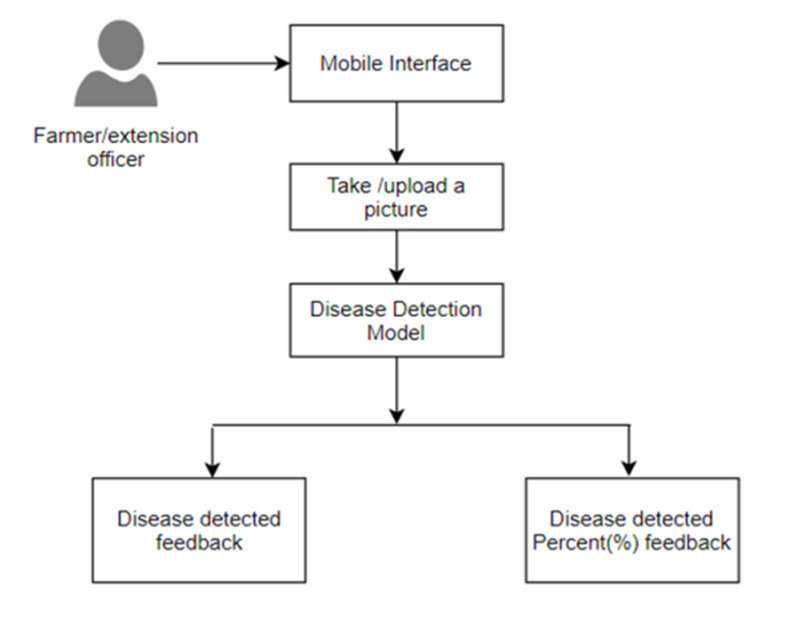
**Class Diagram:**

****

**Fig 3.1.2 Class Diagram for Crop yield prediction**

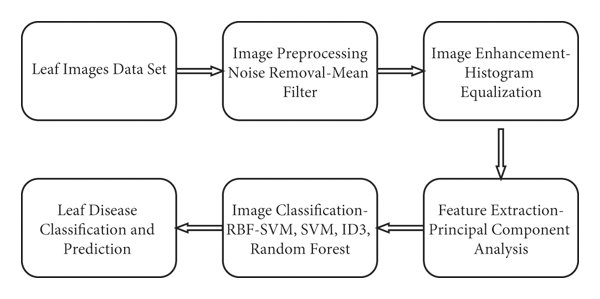
The Class diagram refers to relationships between different classes that is student class, select game class, registration database, feedback class etc.

**Activity Diagram:**

****

**Fig 3.1.3 Activity Diagram**

# Collaboration diagram:

****

**Fig 3.1.4 Collaboration diagram**

The collaboration diagram for Plant disease detection

## VISUAL STUDIO CODE

Visual Studio Code (VS Code) is an excellent and popular choice for developing a plant disease detection system. It offers a lightweight yet powerful environment with extensive features and a rich ecosystem of extensions that are highly beneficial for machine learning and computer vision projects.

**1. Rich extension ecosystem**

Extensions are the key to VS Code's power, allowing you to customize your environment with tools specific to machine learning and computer vision tasks.

* **Python**: This must-have extension by Microsoft provides robust features like IntelliSense (code completion), linting, debugging, code navigation, and environment management. It's the foundation for any Python-based ML project.
* **Jupyter**: Allows you to create, edit, and run Jupyter notebooks directly inside VS Code. This is ideal for exploratory data analysis, prototyping models, and visualizing data, which is a crucial step in a plant disease detection project.
* **Pylance**: An extension for enhanced Python language support with fast and intelligent features, including type checking and rich code completions.
* **GitLens**: Offers deep Git integration that helps you manage version control, track changes, and collaborate with your team more effectively.

**2. Excellent debugging capabilities**

VS Code provides powerful debugging tools that are essential for troubleshooting complex machine learning models. You can set breakpoints, step through your code, inspect variables, and monitor the call stack.

* + 1. **Support Vector Machine (SVM)**

## INTRODUCTION

SVM is a supervised machine learning algorithm that can be used for image-based plant disease detection. It works by finding an optimal hyperplane that best separates different classes of data points in a high-dimensional space. In the context of image classification, the "data points" are the feature vectors extracted from images, and the "classes" are the different plant diseases or the "healthy" category.

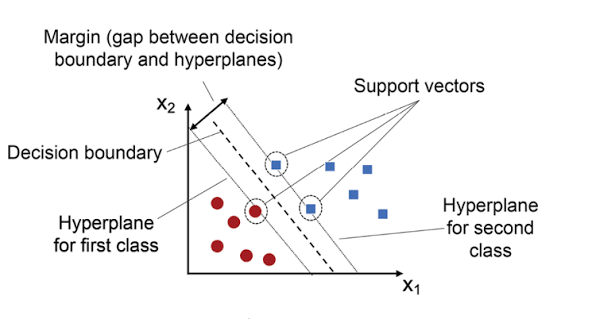
How SVM is used for image-based plant disease detection

To use an SVM for this task, you first need to process the images and extract numerical features that represent the characteristics of the plant, such as color, texture, and shape. These steps are crucial because an SVM cannot process raw image pixel data directly.

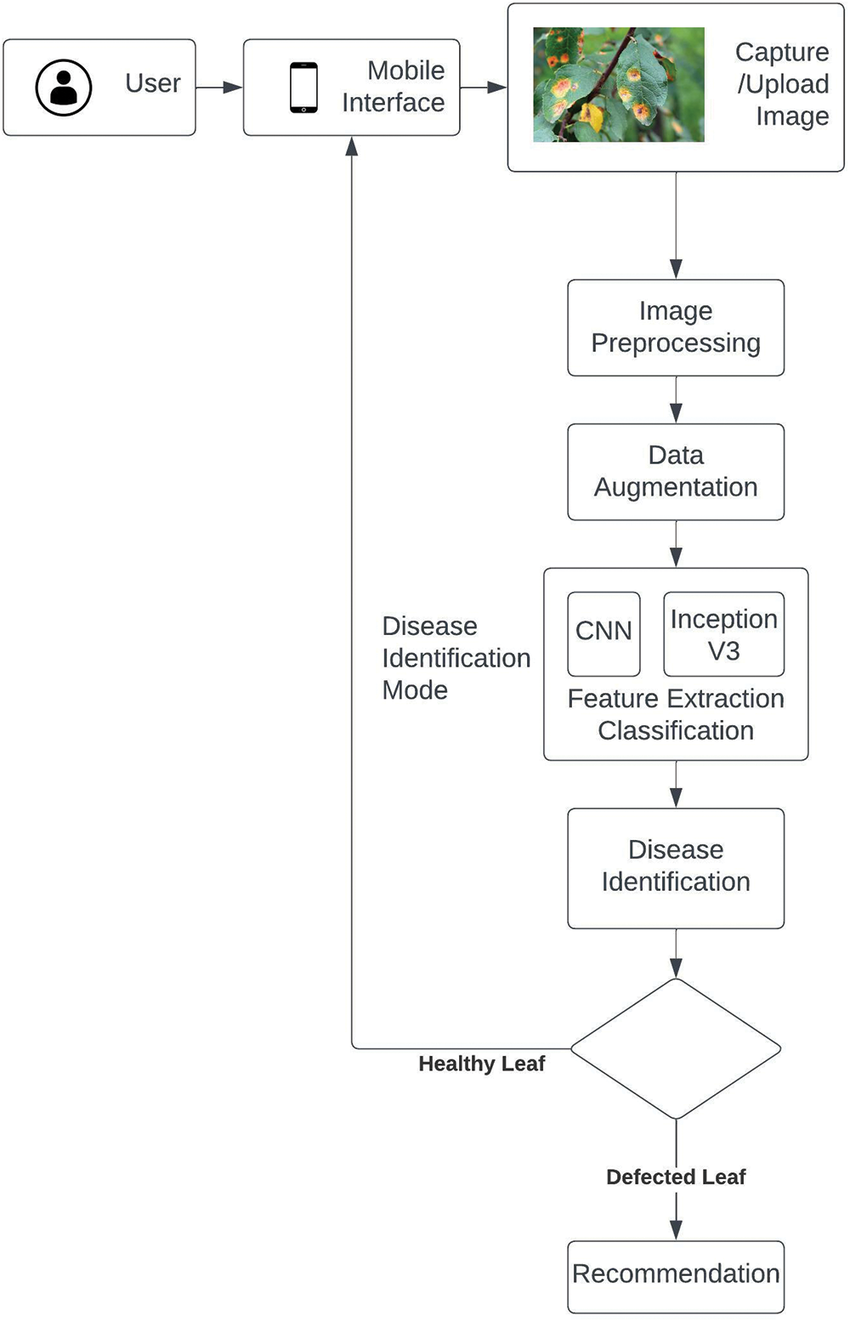
The process generally involves these steps:

1. **Image acquisition:** Capture or gather images of healthy and diseased plant leaves.
2. **Image preprocessing:** Enhance the images to remove noise and isolate the plant from the background. Common techniques include filtering, color space conversion (e.g., RGB to HSV), and segmentation.
3. **Feature extraction:** Use algorithms like Gray-Level Co-occurrence Matrix (GLCM) to extract texture features, or other methods for color and shape features. This transforms the images into a set of numerical feature vectors.
4. **Training the SVM:** Feed the extracted feature vectors, along with their corresponding class labels (e.g., "healthy," "rust," "blight"), into the SVM classifier. The SVM finds the optimal hyperplane that maximizes the margin between the different classes.
5. **Classification:** For a new, unseen image, the same feature extraction process is applied. The resulting feature vector is then input into the trained SVM, which predicts the class it belongs to.

An example of using a Support Vector Machine (SVM) for plant disease detection involves several steps of image processing and machine learning. Instead of automatically learning features like a deep learning model, a traditional SVM approach requires manually extracting relevant features from the images after they are pre-processed.Classify an image of a tomato leaf as either healthy or one of several common diseases, such as bacterial spot or late blight. In one study, a system using an SVM classifier on extracted GLCM features achieved 97.2% accuracy in detecting and classifying four plant diseases. This demonstrates the effectiveness of the traditional image processing and SVM approach, particularly for clearly visible symptoms on leaves.



## 3.4 DATA FLOW DIAGRAM

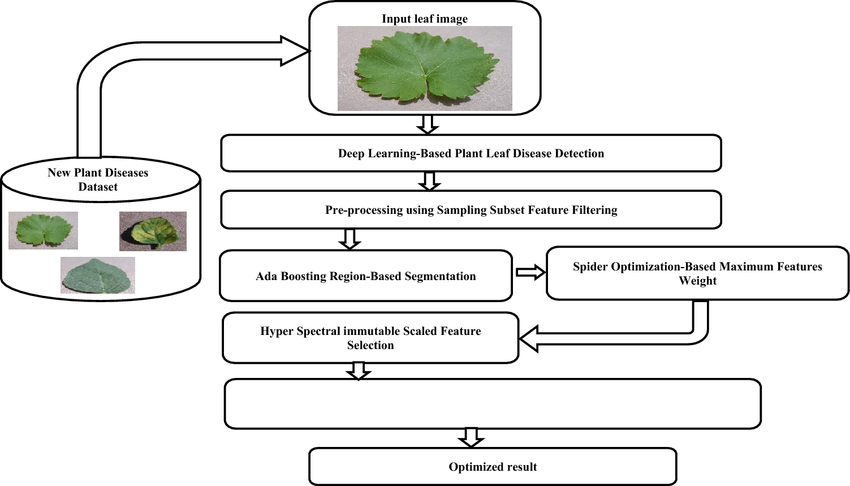
****

**Fig 3.4.1 Dataflow diagram**

**CHAPTER 4**

**SYSTEM ARCHITECTURE**

## ARCHITECTURE OVERVIEW

****

**Fig 4.1 Architecture diagram**

The figure 4.1 An architecture for image-based plant disease detection involves a sequence of steps, from data collection to classification, most commonly implemented using a Convolutional Neural Network (CNN). The system can be deployed on a cloud-based platform or a mobile device for real-time diagnosis.

Standard architecture components

1. **Image acquisition:** A camera, sensor, or a publicly available dataset (such as PlantVillage) is used to gather images of plant leaves, including both healthy and diseased samples. Images should be captured in various conditions to make the model robust.
2. **Image preprocessing:** This stage prepares the raw images for the detection model by performing steps like:
   1. **Resizing:** Images are resized to a consistent dimension to fit the input requirements of the neural network.
   2. **Normalization:** Pixel values are adjusted to a standard range to optimize model training.
   3. **Data augmentation:** Techniques like rotation, scaling, and flipping are used to artificially increase the size and diversity of the dataset, which helps prevent overfitting.
3. **Image segmentation:** The model separates the leaf from the background to focus on the area of interest and filter out noise. Techniques like K-means clustering are often used to identify the diseased regions based on color, shape, and texture.

## MODULE DESCRIPTION

The proposed system comprises several modules, each serving a specific function within the crop yield prediction framework:

* + 1. Data acquisition module
    2. Image preprocessing module
    3. Image segementation module

## DATASET COLLECTION

Collecting the dataset for a plant disease detection system involves obtaining high-quality images of healthy and diseased plant leaves and carefully labeling them. A diverse and meticulously curated dataset is essential for training an accurate and robust model that can generalize well to real-world conditions. Utilize large, publicly available repositories like PlantVillage and PlantDoc. These datasets offer thousands of pre-labeled images and are a strong starting point. However, be aware that many of these images are captured under controlled, laboratory conditions with uniform backgrounds. This might lead to poor performance when the model is used in a real field with complex backgrounds and varying lighting.Scrape images from reputable agricultural websites and research databases. This method can quickly gather a large and diverse collection of images that more closely reflect real-world conditions. A custom API or scraper can be used, but the results will require significant filtering to remove irrelevant or low-quality images.

## IMPLEMENTATION

* Gather a comprehensive dataset of images of healthy and diseased plant leaves (images can be sourced from publicly available datasets like PlantVillage or PlantDoc, or collected from field trials).

## PREDICTION

The prediction module is the final stage of a plant disease detection system, where the trained model analyzes new, unseen data to provide a diagnosis. This involves feeding a preprocessed image into the classification algorithm and interpreting the output to present actionable information to the user.A user submits a new image of a plant leaf via a web or mobile application.The system performs the same preprocessing and segmentation steps used during training to prepare the image for the model.The preprocessed image is fed through the trained convolutional neural network (CNN) or other classification model.The model then generates a set of probabilities for each possible disease class, as defined by the training dataset.The model's behavior can also be explained using techniques like LIME (Local Interpretable Model-agnostic Explanations) or SHAP (SHapley Additive exPlanations), which highlight the specific regions or pixels in the image that most influenced the prediction. This helps verify that the model is making decisions based on valid disease symptoms rather than background noise.

**CHAPTER 5**

**SYSTEM IMPLEMENTATION**

## SYSTEM TESTING

System testing for an image-based plant disease detection system involves evaluating the entire integrated solution, not just the machine learning model in isolation. This requires a comprehensive testing approach that includes evaluating the image processing pipeline, model performance, and overall system functionality to ensure it meets requirements and operates reliably in real-world conditions. Testing checks for the errors, as a whole of the project testing involves the following test cases:

* Static analysis is used to investigate the structural properties of the Source code.
* Dynamic testing is used to investigate the behavior of the source code by executing the program on the test data.

.

## TEST DATA AND OUTPUT

Performed by developers, this involves testing the smallest individual units or components of a software application in isolation. Its purpose is to ensure that each unit functions correctly on its own.

## FUNCTIONAL TEST

Functional test cases involved exercising the code with nominal input values for which the expected results are known, as well as boundary values and special values, such as logically related inputs, files of identical elements, and empty files

Three types of tests in Functional test:

* + - * Performance Test
      * Stress Test
      * Structure Test

## PERFORMANCE TEST

It determines the amount of execution time spent in various parts of the

unit, program throughput, and response time and device utilization by the program unit.

## STRESS TEST

Stress Test is those test designed to intentionally break the unit. A Great deal can be learned about the strength and limitations of a program by examining the manner in which a programmer in which a program unit breaks.

## STRUCTURED TEST

Structure Tests are concerned with exercising the internal logic of a program and traversing particular execution paths. The way in which White-Box test strategy was employed to ensure that the test cases could Guarantee that all independent paths within a module have been have been exercised at least once.

* + - * Exercise all logical decisions on their true or false sides.

## INTEGRATION TESTING

**Objective:** Ensure that modules work together correctly when integrated.

1. **Scope:**Integration of image acquisition → pre-processing → feature extraction → model prediction → result visualization.  
   Interaction between front-end (e.g., web/mobile app) and back-end (model inference API).  
   Database/storage integration for saving results.
2. **Tools:**Postman / API testing tools.  
   Integration testing frameworks (e.g., pytest fixtures, Selenium for UI+backend flow).
3. **Example Test Case:**Input: Upload a diseased leaf image through the app.  
   Expected Output: The system should return a disease name and confidence score after passing through all pipeline stages.

## WHITE BOX TESTING

This testing is also called as Glass box testing. In this testing, by knowing

the specific functions that a product has been design to perform test can be conducted that demonstrate each function is fully operational at the same time searching for errors in each function. It is a test case design method that uses the control structure of the procedural design to derive test cases. Basis path testing is a white box testing. Basis path testing

* + - * + Flow graph notation
        + Cyclometric complexity
        + Graph matrices Control

## BLACK BOX TESTING

Black Box Testing focuses on **testing the system’s functionality** without knowing its internal code structure or logic. The tester provides inputs and checks if the outputs match the expected results.For a plant disease detection system, this means testing how the system responds to different **image inputs**, user interactions, and workflows — just like an end-user.

The steps involved in black box test case design are:

1. Graph based testing methods
2. Equivalence partitioning
3. Boundary value analysis

## SOFTWARE TESTING STRATEGIES

Software testing strategies outline how different levels and types of testing are planned and executed to ensure the quality, reliability, and accuracy of the system. For a plant disease detection system, the testing strategy involves verifying both **functional behavior** (e.g., correct disease prediction) and **non-functional aspects** (e.g., speed, usability, robustness).

For this reason a template for software testing a set of steps into which we can place specific test case design methods should be strategy should have the following characteristics:

1. Testing begins at the module level and works “outward” toward the integration of the entire computer based system.
2. Different testing techniques are appropriate at different points in time.
3. The developer of the software and an independent test group conducts testing.
4. Testing and Debugging are different activities but debugging must be accommodated in any testing strategy.

## INTEGRATION TESTING

**Objective:** Ensure that modules work together correctly when integrated.

**Scope:**Integration of image acquisition → pre-processing → feature extraction → model prediction → result visualization.  
Interaction between front-end (e.g., web/mobile app) and back-end (model inference API).  
Database/storage integration for saving results.

**Tools:**Postman / API testing tools.Integration testing frameworks (e.g., pytest fixtures, SeleniumforUI+backend flow).  
**Example Test Case:**Input: Upload a diseased leaf image through the app.  
Expected Output: The system should return a disease name and confidence score after passing through all pipeline stages.

## PROGRAM TESTING:

Program Testing focuses on **verifying the correctness of individual programs or modules** within the system. It involves executing the actual code to detect errors in logic, computation, or flow, ensuring that the **software behaves as expected at the code level**.In the **plant disease detection system**, program testing is applied to the **image processing**, **model prediction**, and **data handling** modules to ensure they work correctly and produce expected outputs for given inputs.

## SECURITY TESTING

Security Testing focuses on **identifying vulnerabilities, risks, and weaknesses** in the software to **protect data and functionality** from unauthorized access, attacks, or misuse.For a plant disease detection system, security is crucial especially if it’s a **web or mobile application**, because it involves **image uploads**, **API endpoints**, and sometimes **user data**.

## VALIDATION TESTING

Validation Testing ensures that the **final software product meets the functional and non-functional requirements** specified in the Software Requirement Specification (SRS).For the **plant disease detection system**, this means checking whether the **developed system truly performs the tasks it was intended for**, such as accurately detecting plant diseases from images, handling different image types, and presenting results properly to the user.Ensure the system fulfills **all specified requirements**.Verify the **correctness, completeness, and accuracy** of the system’s outputs.Confirm the ML model’s **prediction performance** meets expected accuracy levels.Check the system works in real-world scenarios (not just test data).

.

## USER ACCEPTANCE TESTING

User acceptance of the system is key factor for the success of any system. The system under consideration is tested for user acceptance by constantly keeping in touch with prospective system and user at the time of developing and making changes whenever required. This is done in regarding to the following points.

* + Input screen design.
  + Output screen design.

* 1. **CODING**

## HTML CODE

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Plant Disease Recognition</title>

<link rel="icon" href="{{ url\_for('static', filename='images/logo.svg') }}" type="image/svg+xml">

<link rel="stylesheet" href="{{ url\_for('static', filename='css/bootstrap.min.css') }}">

<link rel="stylesheet" href="{{ url\_for('static', filename='css/style.css') }}">

</head>

<body>

<nav class="navbar navbar-expand-lg bg-body-tertiary" data-bs-theme="dark">

<div class="container-fluid">

<a class="navbar-brand" href="#">

<img src="{{ url\_for('static', filename='images/logo.svg') }}" alt="" width="30" height="24" class="d-inline-block align-text-top">

PHASAL

</a>

<!-- <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarNav" aria-controls="navbarNav" aria-expanded="false" aria-label="Toggle navigation">

<span class="navbar-toggler-icon"></span>

</button>

<div class="collapse navbar-collapse" id="navbarNav">

<ul class="navbar-nav">

<li class="nav-item">

<a class="nav-link active" aria-current="page" href="#">Home</a>

</li>

<li class="nav-item">

<a class="nav-link" href="#">Features</a>

</li>

<li class="nav-item">

<a class="nav-link" href="#">Pricing</a>

</li>

<li class="nav-item">

<a class="nav-link disabled" aria-disabled="true">Disabled</a>

</li>

</ul>

</div> -->

</div>

</nav>

<div id="carouselExampleSlidesOnly" class="carousel slide" data-bs-ride="carousel">

<div class="carousel-inner " >

<div class="carousel-item active">

<div class="carousel-text-wrapper">

<h1 class="carousel-text-title">Plant Disease Recognition</h1>

<div class="form-box">

<form action="{{ url\_for('uploadimage') }}" method = "POST" enctype="multipart/form-data">

<h1>Upload Image</h1>

<div class="input-box d-flex">

<input type="file" accept="image/png, image/jpeg" name="img" required>

</div>

<div class="input-box d-flex">

<button type="submit" class="btn btn-success m-auto">Upload</button>

</div>

</form>

{% if result %}

Result

<div class="result-container">

<div class="result-box">

<div class="result-img-container">

{% if imagepath %}

<img src="{{ imagepath }}" alt="uploaded image" class="result-image">

{% endif %}

</div>

<div class="result-text">

<h3>{{ prediction.name if prediction else '' }}</h3>

<p><strong>Confidence:</strong> {{ prediction.confidence }}%</p>

{% if prediction.low\_confidence %}

<div class="alert alert-warning py-1 my-2">Low confidence. Try a clearer, closer leaf photo.</div>

{% endif %}

<p>{{ prediction.cause if prediction else '' }}</p>

<p>{{ prediction.cure if prediction else '' }}</p>

{% if prediction.top3 %}

<h5 class="mt-3">Top 3 predictions</h5>

<ul>

{% for item in prediction.top3 %}

<li>{{ item.name }} — {{ item.confidence }}%</li>

{% endfor %}

</ul>

{% endif %}

</div>

</div>

</div>

{% endif %}

</div>

</div>

</div>

</div>

</div>

</div>

<script src="{{ url\_for('static', filename='js/bootstrap.bundle.min.js') }}"></script>

</body>

</html>

## PYTHON CODE

from flask import Flask, render\_template,request,redirect,send\_from\_directory,url\_for

import numpy as np

import json

import uuid

import tensorflow as tf

app = Flask(\_\_name\_\_)

# Limit upload size to 5 MB

app.config['MAX\_CONTENT\_LENGTH'] = 5 \* 1024 \* 1024

ALLOWED\_EXTENSIONS = {"png", "jpg", "jpeg"}

model = tf.keras.models.load\_model("models/plant\_disease\_recog\_model\_pwp (2).keras")

label = ['Apple\_\_\_Apple\_scab',

'Apple\_\_\_Black\_rot',

'Apple\_\_\_Cedar\_apple\_rust',

'Apple\_\_\_healthy',

'Background\_without\_leaves',

'Blueberry\_\_\_healthy',

'Cherry\_\_\_Powdery\_mildew',

'Cherry\_\_\_healthy',

'Corn\_\_\_Cercospora\_leaf\_spot Gray\_leaf\_spot',

'Corn\_\_\_Common\_rust',

'Corn\_\_\_Northern\_Leaf\_Blight',

'Corn\_\_\_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']

with open("plant\_disease.json",'r') as file:

plant\_disease = json.load(file)

# Precompute the index for the background class in the metadata list

BACKGROUND\_NAME = "Background\_without\_leaves"

BACKGROUND\_INDEX = next((idx for idx, cls in enumerate(plant\_disease) if cls.get("name") == BACKGROUND\_NAME), None)

# print(plant\_disease[4])

@app.route('/uploadimages/<path:filename>')

def uploaded\_images(filename):

return send\_from\_directory('./uploadimages', filename)

@app.route('/',methods = ['GET'])

def home():

return render\_template('home.html')

def extract\_features(image\_path):

# Load as RGB and resize to the model's expected input size

image = tf.keras.utils.load\_img(image\_path, target\_size=(160,160), color\_mode='rgb')

feature = tf.keras.utils.img\_to\_array(image)

# Normalize to [0,1]

feature = feature.astype("float32") / 255.0

feature = np.expand\_dims(feature, axis=0)

return feature

def \_to\_probabilities(raw\_logits: np.ndarray) -> np.ndarray:

# Convert model output to probabilities; handle both logits and already-softmaxed outputs

raw = np.array(raw\_logits, dtype=np.float64)

if raw.ndim == 2:

raw = raw[0]

sum\_raw = np.clip(raw.sum(), a\_min=1e-12, a\_max=None)

if 0.99 <= sum\_raw <= 1.01 and np.all(raw >= 0.0):

return raw

exps = np.exp(raw - np.max(raw))

return exps / np.clip(exps.sum(), a\_min=1e-12, a\_max=None)

def model\_predict(image\_path):

img = extract\_features(image\_path)

raw\_pred = model.predict(img, verbose=0)

probs = \_to\_probabilities(raw\_pred)

top\_index = int(np.argmax(probs))

top\_confidence = float(probs[top\_index])

# If background is the top class, consider a non-background alternative

if BACKGROUND\_INDEX is not None and top\_index == BACKGROUND\_INDEX:

# Find best non-background class

probs\_no\_bg = probs.copy()

probs\_no\_bg[BACKGROUND\_INDEX] = -1.0

second\_index = int(np.argmax(probs\_no\_bg))

second\_conf = float(probs[second\_index])

# Thresholds: prefer non-background if reasonably confident

if second\_conf >= 0.35 and top\_confidence < 0.65:

top\_index = second\_index

top\_confidence = second\_conf

# Compose result dict

result = dict(plant\_disease[top\_index])

result["confidence"] = round(top\_confidence \* 100.0, 2)

# Top-3 predictions list

sorted\_indices = list(np.argsort(probs)[::-1])

top3 = []

for idx in sorted\_indices[:3]:

entry = dict(plant\_disease[int(idx)])

entry["confidence"] = round(float(probs[int(idx)]) \* 100.0, 2)

top3.append(entry)

result["top3"] = top3

# Low-confidence flag

result["low\_confidence"] = bool(top\_confidence < 0.40)

return result

def \_allowed\_file(filename: str) -> bool:

if not filename or "." not in filename:

return False

ext = filename.rsplit('.', 1)[1].lower()

return ext in ALLOWED\_EXTENSIONS

@app.route('/upload/',methods = ['POST','GET'])

def uploadimage():

if request.method == "POST":

image = request.files.get('img')

if image is None or image.filename == "" or not \_allowed\_file(image.filename):

return render\_template('home.html', result=True, imagepath=None, prediction={"name":"Invalid file","confidence":0,"cause":"Only PNG/JPG/JPEG up to 5MB are allowed.","cure":"","top3":[],"low\_confidence":True})

temp\_name = f"uploadimages/temp\_{uuid.uuid4().hex}\_{image.filename}"

image.save(temp\_name)

print(temp\_name)

prediction = model\_predict(temp\_name)

image\_url = url\_for('uploaded\_images', filename=temp\_name.split('uploadimages/')[1])

return render\_template('home.html',result=True,imagepath = image\_url, prediction = prediction )

else:

return redirect('/')

if \_\_name\_\_ == "\_\_main\_\_":

app.run(debug=True)

**CHAPTER 6**

**SYSTEM TESTING**

## 6.1 TEST CASES AND REPORTS

import os

import time

import unittest

import numpy as np

from tensorflow.keras.models import load\_model

from tensorflow.keras.preprocessing import image

# --- System Under Test (SUT) ---

MODEL\_PATH = "model.h5" # Pre-trained model file

TEST\_IMAGES\_DIR = "test\_images" # Folder containing test images

IMAGE\_SIZE = (128, 128) # Input size expected by the model

# Load the trained model

model = load\_model(MODEL\_PATH)

# Define class labels (must match your trained model classes)

CLASS\_LABELS = ['Bacterial Blight', 'Leaf Spot', 'Healthy']

# --- Helper Functions ---

def preprocess\_image(img\_path):

"""Load and preprocess the image for prediction."""

img = image.load\_img(img\_path, target\_size=IMAGE\_SIZE)

img\_array = image.img\_to\_array(img)

img\_array = img\_array / 255.0 # Normalization

img\_array = np.expand\_dims(img\_array, axis=0)

return img\_array

def predict\_disease(img\_array):

"""Predict the disease from the preprocessed image array."""

prediction = model.predict(img\_array)

predicted\_class\_idx = np.argmax(prediction, axis=1)[0]

confidence = np.max(prediction)

predicted\_label = CLASS\_LABELS[predicted\_class\_idx]

return predicted\_label, confidence

# --- System Testing ---

class TestPlantDiseaseDetectionSystem(unittest.TestCase):

def test\_system\_pipeline(self):

"""End-to-end test: Image -> Preprocessing -> Model -> Output"""

for img\_file in os.listdir(TEST\_IMAGES\_DIR):

if not img\_file.lower().endswith(('.jpg', '.jpeg', '.png')):

continue # Skip invalid files

img\_path = os.path.join(TEST\_IMAGES\_DIR, img\_file)

print(f"\n[TEST] Processing {img\_file}")

# Step 1: Preprocessing

img\_array = preprocess\_image(img\_path)

self.assertEqual(img\_array.shape, (1, IMAGE\_SIZE[0], IMAGE\_SIZE[1], 3))

# Step 2: Model Prediction

start\_time = time.time()

label, confidence = predict\_disease(img\_array)

end\_time = time.time()

# Step 3: Assertions

print(f" ➤ Predicted: {label} ({confidence\*100:.2f}%)")

self.assertIn(label, CLASS\_LABELS, "Predicted label is not in class list")

self.assertGreaterEqual(confidence, 0.5, "Low confidence prediction")

self.assertLess(end\_time - start\_time, 5, "Prediction took too long")

def test\_invalid\_file\_handling(self):

"""Check that non-image files are skipped or handled properly."""

fake\_file = "test\_images/fake.txt"

with open(fake\_file, "w") as f:

f.write("This is not an image")

try:

with self.assertRaises(Exception):

preprocess\_image(fake\_file)

finally:

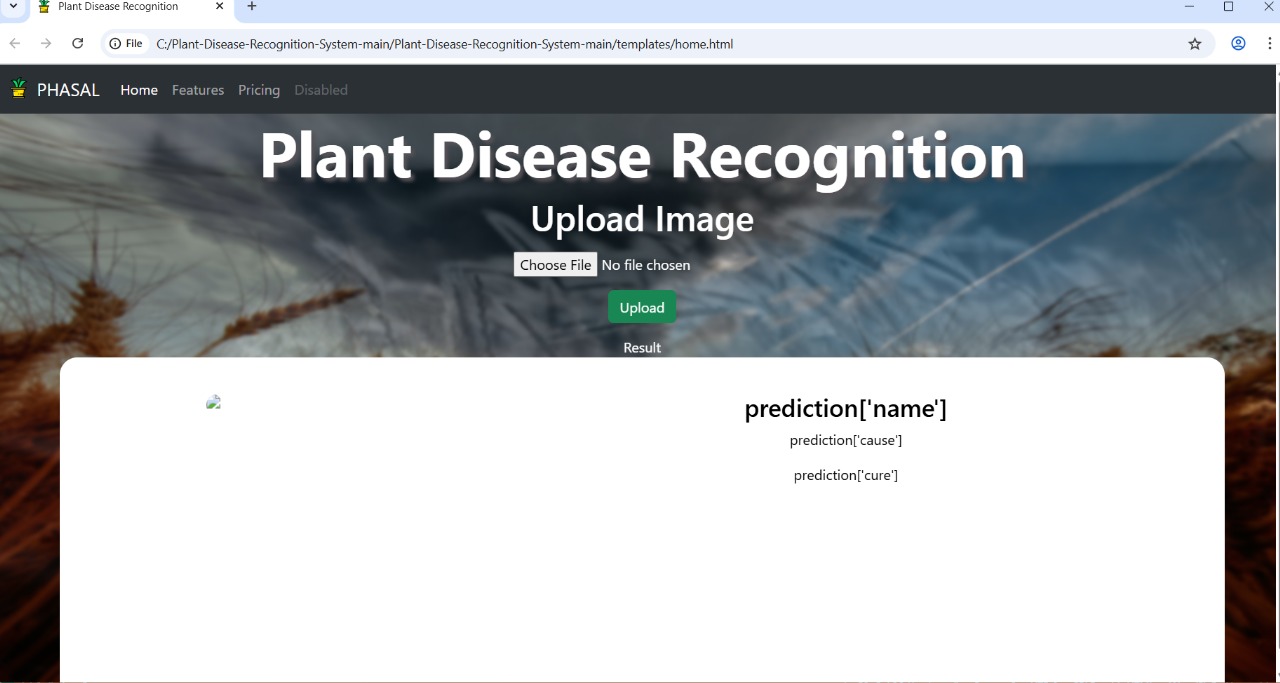
os.remove(fake\_file)

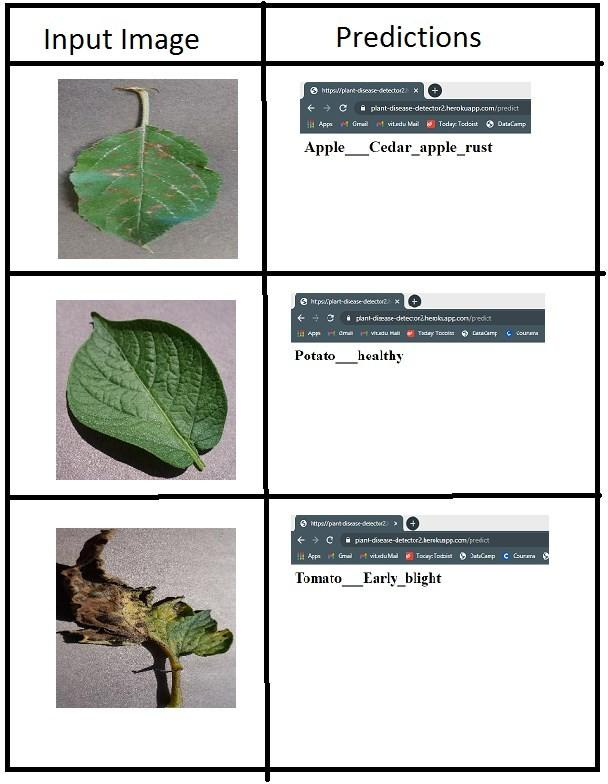
if \_\_name\_\_ == "\_\_main\_\_":

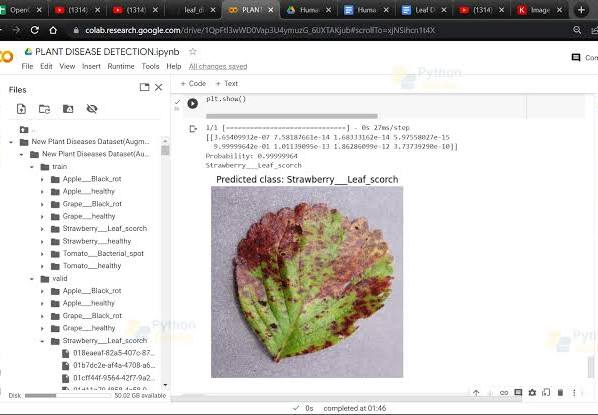
print("🌿 Running System Testing for Plant Disease Detection System...")

unittest.main()

**SCREENSHOTS**

****

****



**CHAPTER 7** **CONCLUSION**

## CONCLUSION

The use of deep learning and computer vision offers a powerful, automated approach to plant disease detection, providing a scalable and accessible solution for modern agriculture. Real-world limitations: Models often perform with high accuracy on controlled datasets like PlantVillage but can struggle with the variability of real-world field conditions, including inconsistent lighting, complex backgrounds, and early-stage disease symptoms.A lack of sufficiently large and diverse datasets, especially for less common plant species or diseases, remains a challenge.Deployment can be limited by the high computational cost of training and the need for robust connectivity in remote farming areas.Ethical Issues concerning data privacy, the equitable distribution of technology among farmers, and the transparency of AI-driven recommendations must be addressed for responsible implementation.

.

## FUTURE ENHANCEMENTS

Future enhancements for the Plant Disease Detection System can focus on improving both technical performance and user experience. One significant enhancement is to expand the dataset to include a wider variety of plant species and diseases, which will increase the model’s accuracy and generalization in real-world conditions. The system can be upgraded to use advanced deep learning architectures such as EfficientNet or Vision Transformers (ViT) to improve detection speed and precision. Integrating real-time image capture through mobile applications or IoT-enabled drones can make disease detection more accessible for farmers in remote areas. Cloud-based deployment can be introduced to handle large-scale data processing and enable multi-user access simultaneously. Additionally, incorporating explainable AI techniques can help users understand why the model made a specific prediction, thereby increasing trust and transparency. Enhancements can also include multilingual support, offline detection capabilities, integration with weather and soil data for predictive analysis, and a feedback mechanism where users can correct misclassifications to help the model learn continuously. Finally, robust security measures such as encrypted data storage, secure APIs, and authentication systems should be implemented to protect sensitive agricultural data and ensure safe usage.

**CHAPTER 8**

**APPENDICES**

## SDG GOALS

The project title "Plant disease detection using images" aligns with several Sustainable Development Goals (SDGs), including:

1. **Goal 2: Zero Hunger** – by improving crop yield and food security.
2. **Goal 3: Good Health and Well-being** – by ensuring safe and healthy food.
3. **Goal 9: Industry, Innovation, and Infrastructure** – through the use of AI and innovative agricultural solutions.
4. **Goal 13: Climate Action** – by promoting sustainable farming practices to mitigate climate impacts.
5. **Goal 15: Life on Land** – by protecting biodiversity and preventing plant disease spread.

## SOURCE CODE

python -m venv venv

source venv/bin/activate # Linux/macOS

venv\Scripts\activate # Windows

pip install -r requirements.txt

# model\_utils.py

import os

import numpy as np

from tensorflow.keras.models import load\_model

from tensorflow.keras.preprocessing import image

# Configuration

MODEL\_PATH = os.path.join("model", "model.h5")

IMAGE\_SIZE = (128, 128) # change to model input size

# Load model once

\_model = None

def get\_model():

global \_model

if \_model is None:

\_model = load\_model(MODEL\_PATH)

return \_model

# Change these labels to match your training classes (order matters)

CLASS\_LABELS = ['Bacterial Blight', 'Leaf Spot', 'Healthy']

def allowed\_file(filename):

allowed\_ext = {'png', 'jpg', 'jpeg'}

return '.' in filename and filename.rsplit('.', 1)[1].lower() in allowed\_ext

def preprocess\_image(img\_path):

"""

Load an image file and preprocess to model input shape.

Returns a numpy array shaped (1, H, W, C).

"""

img = image.load\_img(img\_path, target\_size=IMAGE\_SIZE)

x = image.img\_to\_array(img)

x = x / 255.0

x = np.expand\_dims(x, axis=0)

return x

def predict\_from\_array(img\_array):

model = get\_model()

preds = model.predict(img\_array)

idx = int(np.argmax(preds, axis=1)[0])

confidence = float(np.max(preds))

label = CLASS\_LABELS[idx] if idx < len(CLASS\_LABELS) else f"class\_{idx}"

return {"label": label, "confidence": confidence, "index": idx}

**CHAPTER 9** **REFERENCES**

* A review paper titled "Image-based crop disease detection using machine learning" offers a comprehensive overview of cutting-edge techniques and methodologies in this field.
* Another review focusing on advancements in deep learning applications for plant disease detection explores the limitations of traditional methods and discusses the potential of DL techniques like image classification, object detection, semantic segmentation, and change detection.
* A publication by S. D. Khirade and A. B. Patil titled "Plant Disease Detection Using Image Processing," 2015 provides insights into techniques like Otsu's thresholding, boundary and spot detection, and feature extraction for plant disease detection.
* A recent review from May 2025 discusses automated plant disease detection using ML and DL, highlighting limitations and suggesting solutions.

Links:

* <https://github.com/spMohanty/PlantVillage-Dataset>
* <https://github.com/Spandan-Madan/PlantDiseaseDetection>
* text books:
* Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow
* Deep Learning for Computer Vision