Plant Detection Using Streamlit

Mini Project-I

(ENSI152)

B.Tech CSE CORE (SEC-B)

to

K.R Mangalam University

by

Vanadana Sharma (2401010285)
SAKSHI GARG (2401010178)
POOJA AGGARWAL (2401010240)
JYOTI SURYAVANSHI (2401010291)

Under the supervision of

Dr.Pankaj Aggarwal
Dean OF SOET
K.R Mangalam University

Department of Computer Science and Engineering

School of Engineering and Technology

K.R Mangalam University, Gurugram- 122001, India

April 2025

Index

- 9.1 Technical Challenges
- 9. 2 Development Challenges
- 9. 3 Learning Curve

Chapter 10: Future Enhancements

- 10.1 Offline Mode
- 10.2 Plant Care Recommendations
- 10.3 Support for More Plant Species
- 10.4 Integration with Agriculture Platforms
- 10.5 User Account and History
- 10.6 Multilingual Interface
- 10.7 Real-time Notifications and Alerts

Chapter 11: Conclusion

Chapter 12: Bibliography/References

Chapter 1: Introduction

1.1 Background

Agriculture has always played a critical role in human survival, development, and economic growth. In many countries, especially developing nations like India, agriculture contributes significantly to GDP and employment. However, one of the biggest challenges faced by farmers and agricultural scientists is the accurate identification of plant species and detection of crop diseases. Traditionally, plant identification relies on human expertise, which is often limited, especially in rural or underdeveloped areas.

In recent years, Artificial Intelligence (AI) has emerged as a transformative technology in many sectors, including agriculture. AI-powered plant detection systems can process images of plants to recognize species, diagnose diseases, and even provide treatment suggestions. With the growth of smartphone usage and internet accessibility, AI-based plant detection can now be deployed as simple, accessible applications, helping farmers, students, and agricultural researchers alike.

This project leverages AI technologies and the Python-based Streamlit framework to build an easy-to-use web application capable of detecting plant species using either uploaded images or real-time webcam feeds. By integrating computer vision techniques and trained AI models, the application provides fast and accurate plant identification, bridging the gap between modern technology and grassroots-level farming needs.

1.2 Objective

The primary objective of this project is to design and develop a functional and user-friendly application for plant species detection. This application should be:

- Simple to use, requiring no technical expertise.
- Fast in delivering results after the image is provided.
- Accurate, based on a well-trained AI model.
- Accessible, available via a web browser interface.

Specific goals include:

- Enabling users to upload or capture an image of a plant through a webcam.
- Sending this image to an AI model hosted via an inference API.
- Displaying the detection results in real-time, along with visual cues like bounding boxes.
- Creating a Streamlit interface that is visually clean and interactive.
- Building a foundation for future features such as plant disease identification, treatment recommendations, and offline use.

1.3 Problem Statement

Plant identification and health assessment have traditionally required extensive botanical knowledge, which is not always available to the general public or small-scale farmers. This creates several challenges:

- Lack of Expertise: Many users cannot differentiate between plant species, especially during early growth stages or when diseases are present.
- Time-Consuming: Manual identification through books, internet searches, or expert consultations is slow and unreliable.
- Costly Services: Existing apps often have limited free features or require subscriptions, making them less accessible.
- Scalability Issues: Manual processes do not scale well when working with large farms or multiple plant types.

Thus, there is a clear need for a scalable, automated solution that offers real-time plant detection using modern AI technologies. This project seeks to address these issues through an integrated AI + Streamlit-based solution that offers fast, accurate, and user-friendly plant detection to a wide range of users. Manual plant identification is time-consuming and often inaccurate without expert knowledge. There is a need for an automated solution that can:

- Accurately identify plant species
- Provide instant results
- Be easy to use for non-technical users

Thus, there is a clear need for a scalable, automated solution that offers real-time plant detection using modern AI technologies. This project seeks to address these issues through an integrated AI + Streamlit-based solution that offers fast, accurate, and user-friendly plant detection to a wide range of users.

Chapter 2: Literature Review

2.1 Existing Solutions and Their Limitations

Over the past decade, the integration of technology into agriculture has led to the development of various tools for plant species identification. Among the most notable solutions are mobile applications such as PlantNet, PlantSnap, and LeafSnap. These apps typically allow users to upload an image of a plant or leaf, which is then compared against a vast database of images using machine learning algorithms to determine the plant's identity.

Despite their popularity and usefulness, these solutions exhibit several limitations that restrict their broader applicability:

- **Limited Species Database**: Many existing tools are constrained by the scope of their datasets. This means they can accurately identify only a limited range of plant species, often specific to a certain region or botanical category.
- **Heavy Internet Dependence**: A majority of these platforms rely on cloud-based inference. This requires an active and stable internet connection, which is not always available in rural or farming regions where such tools are most needed.
- **Subscription-Based Features**: While initial usage may be free, many applications hide advanced features, such as high-resolution identification or detailed plant information, behind paywalls. This reduces accessibility for students, hobbyists, or small-scale farmers.
- **Lack of Real-Time Feedback**: Some systems take considerable time to process and return results, reducing their utility for on-field or rapid assessments.
- **Opaque Algorithms**: Users often do not get insights into how decisions are made, leading to skepticism and a lack of trust in the system.
- **Limited Integration Capabilities**: Most apps are not open-source, making it challenging for developers to customize or integrate them with other agricultural solutions, such as disease detection systems or farm management software.

Given these limitations, there is a critical need for more accessible, transparent, and extensible tools that use modern AI techniques for plant detection.

2.2 Use of AI in Agriculture

Artificial Intelligence has emerged as a transformative force in agriculture, offering new possibilities in everything from precision farming to crop health analysis. Some of the key areas where AI has made a significant impact include:

- **Image-Based Disease Detection**: Using deep learning algorithms, AI systems can detect diseases in crops by analyzing images of leaves, fruits, or stems. These models can identify subtle patterns that are not visible to the human eye, allowing for early intervention.
- **Automated Pest Detection**: Machine learning models are also employed to identify pest infestations. This allows for timely application of pesticides and minimizes crop loss.
- **Crop Yield Prediction**: AI models can forecast crop production by analyzing environmental factors, soil data, and historical crop yields. This aids in better planning and resource allocation.
- **Precision Agriculture**: AI enables the precise application of water, fertilizers, and pesticides by analyzing data from sensors, drones, and satellites. This reduces waste, increases productivity, and promotes sustainable farming.
- **Weed Detection and Removal**: Computer vision algorithms can differentiate between crops and weeds, enabling automated weeding systems that selectively remove unwanted plants without harming the crop.

In the context of plant detection, AI systems are typically trained using Convolutional Neural Networks (CNNs). These networks are designed to recognize and extract features such as shape, texture, and color patterns from plant images. A well-trained CNN can differentiate between thousands of plant species with high accuracy.

By integrating these models with user-friendly platforms like Streamlit, developers can build interactive web applications that democratize access to this technology. Streamlit allows for rapid deployment and easy interfacing of AI models without needing complex front-end development, making it ideal for educational projects and prototype deployments.

Overall, the application of AI in plant detection represents a powerful convergence of technology and nature, offering scalable and practical solutions for modern agricultural challenges.

Chapter 3: Project Overview

3.1 Purpose of the App

The primary purpose of the Plant Detection App is to leverage artificial intelligence and machine learning technologies to identify plant species using images captured or uploaded by users. With the integration of the Roboflow API and a trained detection model, this Streamlit-based web application provides a seamless platform for plant recognition. It is designed for educational purposes, botanical research, hobby gardening, and agricultural fieldwork.

The app aims to simplify plant identification by eliminating the need for manual searching through plant catalogs or dependence on experts. Instead, users can capture an image of a plant using their webcam or upload an image file, and the app will process the image using an AI model to display the plant's name and other relevant information (if integrated). This real-time capability offers a modern solution to a traditional problem.

3.2 Scope of the Project

This project focuses on developing a minimal yet functional AI-based web app that demonstrates the use of machine learning in practical scenarios. The following outlines the scope of the plant detection app:

- **Platform**: A web-based interface built using Streamlit, ensuring platform independence and ease of access.
- **User Interactions**: The user can upload images from their device or capture them via a webcam. The interface is designed to be simple and intuitive.
- **Plant Detection**: The backend uses a pretrained object detection model hosted on Roboflow. The model is capable of recognizing multiple plant species based on training data.
- **Model Inference**: The application sends images to the inference API and displays bounding boxes around detected plants along with their labels.
- **Extensibility**: While the current version may support a limited number of species, the model can be retrained or fine-tuned for expanded detection. Additional features such as offline detection, disease identification, and integration with a plant care database are also possible in future versions.

- **Educational Impact**: As a B.Tech first-year project, this application illustrates fundamental concepts in Python programming, web development using Streamlit, and AI model deployment.

The project is not intended to be a production-grade tool but rather a prototype demonstrating how AI and modern web technologies can be combined to solve a real-world problem effectively. Future enhancements could transform it into a powerful utility for farmers, researchers, and nature enthusiasts.

Chapter 4: Tools & Technologies

4.1 Python

Python is the backbone of this project, serving as the primary programming language. It is widely used in data science, artificial intelligence, and web development due to its simplicity and readability. Python's vast ecosystem of libraries and frameworks makes it an ideal choice for building machine learning-based applications. In this project, Python is used for writing the application logic, integrating the AI model, handling image processing, and setting up the Streamlit interface.

4.2 Streamlit

Streamlit is an open-source Python framework specifically designed for creating interactive and data-driven web applications quickly. Unlike traditional web frameworks, Streamlit allows developers to build UIs with minimal code and no need for frontend expertise. It supports features such as live image capture, file uploads, sidebar navigation, and real-time updates. For this project, Streamlit enables a smooth user experience by creating an interface where users can upload or capture images, initiate detection, and view results—all from a single web page.

4.3 OpenCV

OpenCV (Open Source Computer Vision Library) is a powerful tool for image processing. In this project, it is used to capture webcam input, resize and preprocess images, and display real-time frames. OpenCV facilitates frame manipulation and transformation, ensuring that the images passed to the detection model meet the required format. Its ability to integrate seamlessly with Python and Streamlit makes it crucial for handling visual data.

4.4 NumPy

NumPy (Numerical Python) is a library that provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on them. In this application, NumPy is used for handling image data arrays, normalizing pixel values, and managing data transformations. Its speed and efficiency make it ideal for processing image data before and after AI model inference.

4.5 Roboflow (Inference API)

Roboflow is a popular platform for training, deploying, and hosting computer vision models. It offers tools for dataset preparation, annotation, model training, and deployment via APIs. This project uses a Roboflow-hosted inference model that takes image input and returns object detection results in the form of bounding boxes and class labels. The integration of Roboflow's inference API allows the application to perform real-time detection without requiring local model hosting, simplifying deployment and maintenance.

Chapter 5: System Architecture

5.1 Frontend & Backend Design

The architecture of the plant detection application is a combination of a lightweight frontend built using Streamlit and a backend powered by Roboflow's hosted inference API. This separation of concerns enables a modular, scalable, and easy-to-maintain system.

Frontend (Client-side):

- Built using Streamlit, the frontend acts as the user interface.
- Users can interact via buttons, file uploaders, webcam capture, and image preview sections.
- It sends images (uploaded or captured) to the backend for inference.
- Results from the backend (bounding boxes and labels) are displayed in real time.

Backend (Server-side/Cloud-based):

- The Roboflow inference API hosts a pre-trained machine learning model.
- When an image is sent from the frontend, the API performs object detection and returns JSON output.
- This response includes plant names, confidence scores, and coordinates for bounding boxes.
- The backend logic in the Python script parses this JSON and overlays results on the image.

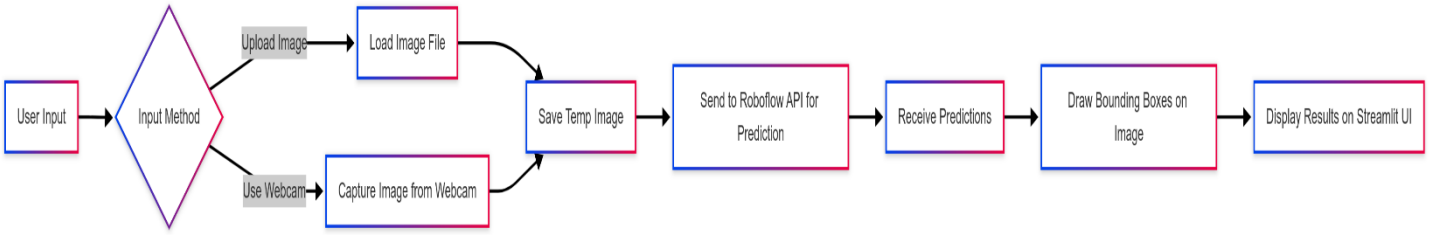
This decoupled architecture ensures the app remains fast and responsive, with heavy computation handled off-device by Roboflow's infrastructure.

5.2 Workflow Diagram

Below is a conceptual description of the system workflow:

1. **User Input**:
    - User uploads an image or captures one using the webcam.
2. **Image Processing**:

    - The image is resized and converted to base64 or suitable format.
    - The image is sent via a POST request to the Roboflow inference API.
3. **Model Inference**:
    - The cloud-hosted model analyzes the image and detects plants.
    - It returns results in JSON containing coordinates and labels.
4. **Display Results**:
    - The frontend overlays bounding boxes and labels on the image.
    - The final annotated image is shown to the user.



Chapter 6: Methodology

6.1 Step-by-Step Process of Implementation

The plant detection application was developed using a structured and iterative methodology that included planning, design, development, testing, and refinement. Below is a detailed explanation of the key implementation steps:

Step 1: Requirement Analysis

- Defined the goal: develop a web application to detect plant species using images.
- Identified tools and platforms suitable for a beginner-level project (e.g., Python, Streamlit, Roboflow).

Step 2: Designing the Interface

- Used Streamlit components to design the UI.
- Incorporated buttons, uploaders, webcam integration, and image display windows.

Step 3: Preparing the AI Model

- Selected a pre-trained model from Roboflow for plant detection.
- Analyzed its dataset to ensure it was capable of identifying a diverse set of plant species.

Step 4: Integrating the Model with the App

- Integrated the Roboflow inference API into the Python code.
- Developed functions to encode images, send them to the API, and parse JSON results.

Step 5: Processing and Displaying Output

- Used OpenCV to draw bounding boxes and annotate detected plants.
- Displayed the processed image in the Streamlit app.

Step 6: Testing and Debugging

- Conducted repeated tests with different image types (bright, blurry, different species).
- Fine-tuned the preprocessing steps and response rendering.

6.2 Data Flow & Detection Process

The data flow in the app follows a linear pipeline model. Here is how the detection process works internally:

1. **Image Acquisition**:
    - The user either uploads an image file or captures one using the webcam integrated into the Streamlit interface.
2. **Preprocessing**:
    - The captured/uploaded image is resized to fit the model's input dimensions.
    - Image is converted into a base64 format or binary format for transmission.
3. **Inference**:
    - The image is sent to the Roboflow model endpoint using an HTTP POST request.
    - The model returns predictions in JSON format including class labels and bounding box coordinates.
4. **Result Processing**:
    - JSON data is parsed in Python.
    - OpenCV is used to draw boxes and labels on the image.
5. **Output Display**:
    - The final annotated image is displayed using Streamlit's image viewer component.

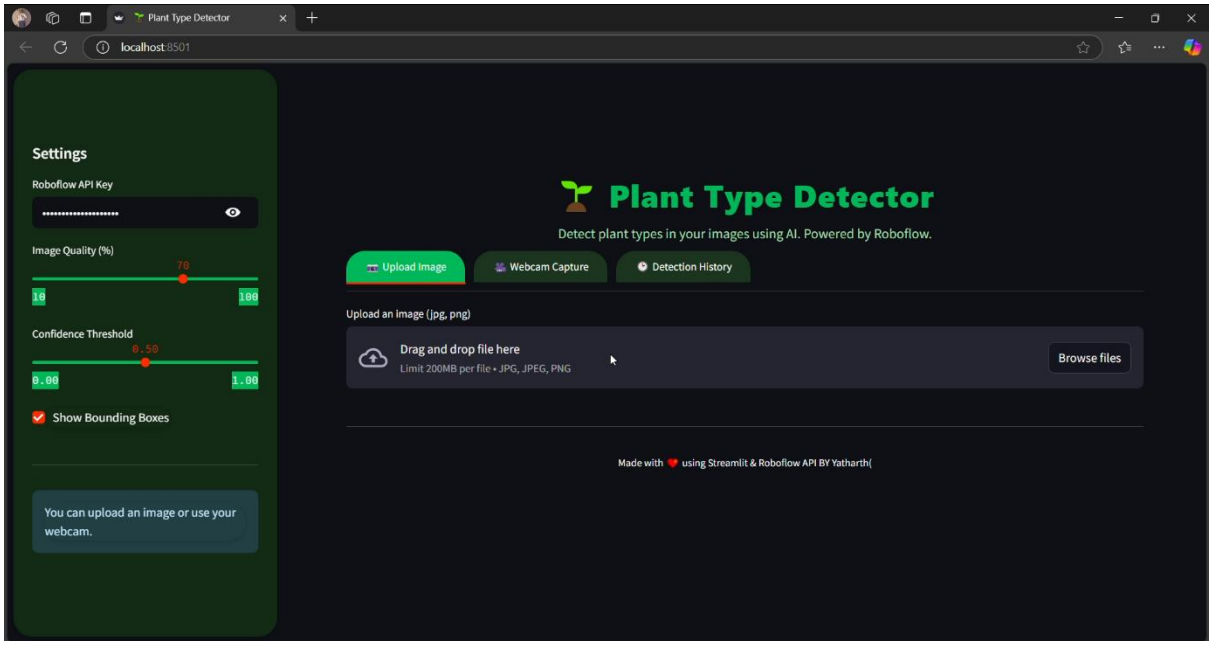
Chapter 7: User Interface

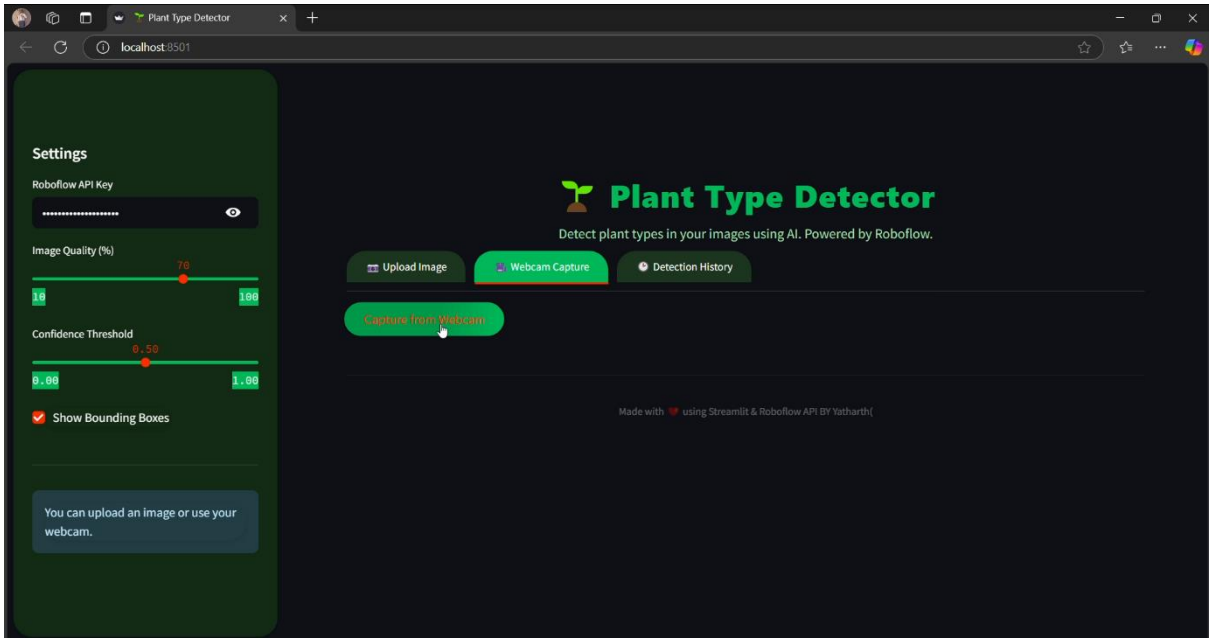
7.1 Streamlit Layout

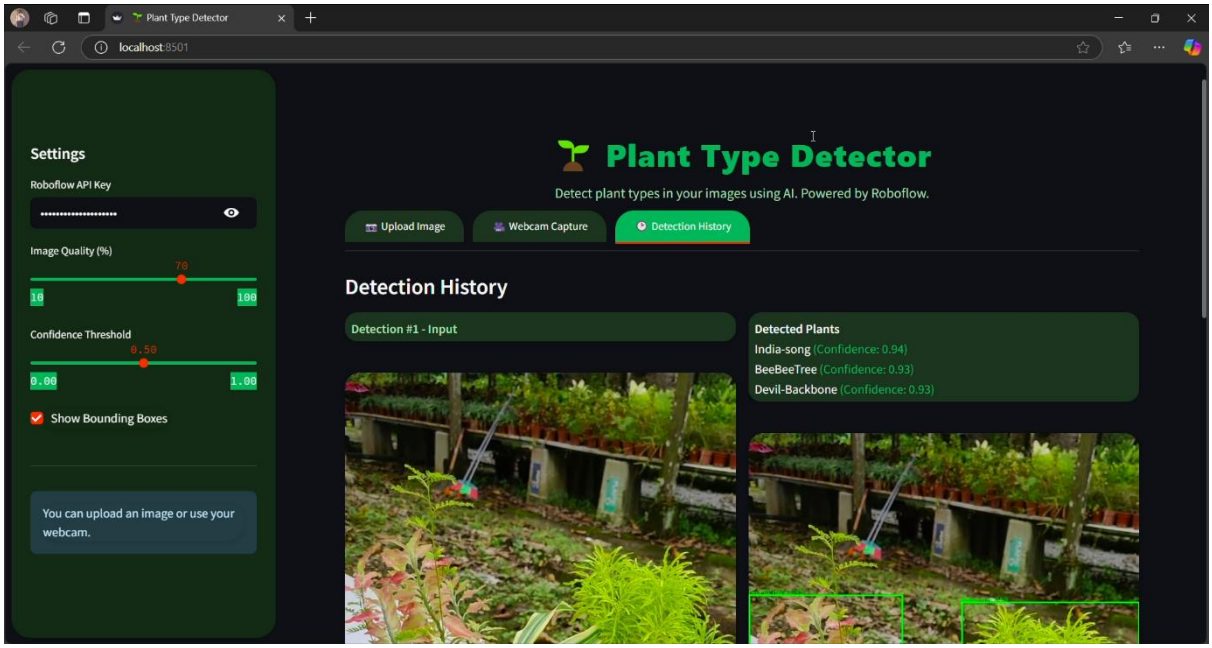
The user interface (UI) of the plant detection application is designed using Streamlit, which offers a clean and intuitive web-based layout. The main layout is divided into sections that help the user to interact with the application in a guided and efficient manner. The app layout is minimal yet functional, enabling easy operation even for non-technical users.

Main Components of the Layout:

- **Title and Description:** At the top of the interface, a title header such as “Plant Detection App” is displayed using st.title(). A brief description or instructions are added below using st.markdown().
- **Image Upload Section:** Users can upload images using st.file_uploader(), which allows common formats like JPEG, PNG, etc.
- **Webcam Capture Section:** For real-time plant detection, a webcam module is integrated using OpenCV and custom functions.
- **Detection Trigger:** A detection button allows users to send the image to the backend model.
- **Results Display:** Annotated results with bounding boxes and labels are displayed using st.image().
- **Sidebar (Optional):** Some versions include a sidebar using st.sidebar for additional settings like confidence thresholds.




Plant Type Detector
localhost:8501
Settings
Roboflow API Key
Image Quality (%)
70
10
100
Confidence Threshold
0.50
0.00
1.00
Show Bounding Boxes
You can upload an image or use your webcam.
Plant Type Detector
Detect plant types in your images using AI. Powered by Roboflow.
Upload Image
Webcam Capture
Detection History
Capture from Webcam
Made with using Streamlit & Roboflow API BY Yatharth(


Plant Type Detector
localhost:8501
Settings
Roboflow API Key
Image Quality (%)
70
10
100
Confidence Threshold
0.50
0.00
1.00
Show Bounding Boxes
You can upload an image or use your webcam.
Plant Type Detector
Detect plant types in your images using AI. Powered by Roboflow.
Upload Image
Webcam Capture
Detection History
Detection History
Detection #1 - Input
Detected Plants
India-song (Confidence: 0.94)
BeeBeeTree (Confidence: 0.93)
Devil-Backbone (Confidence: 0.93)

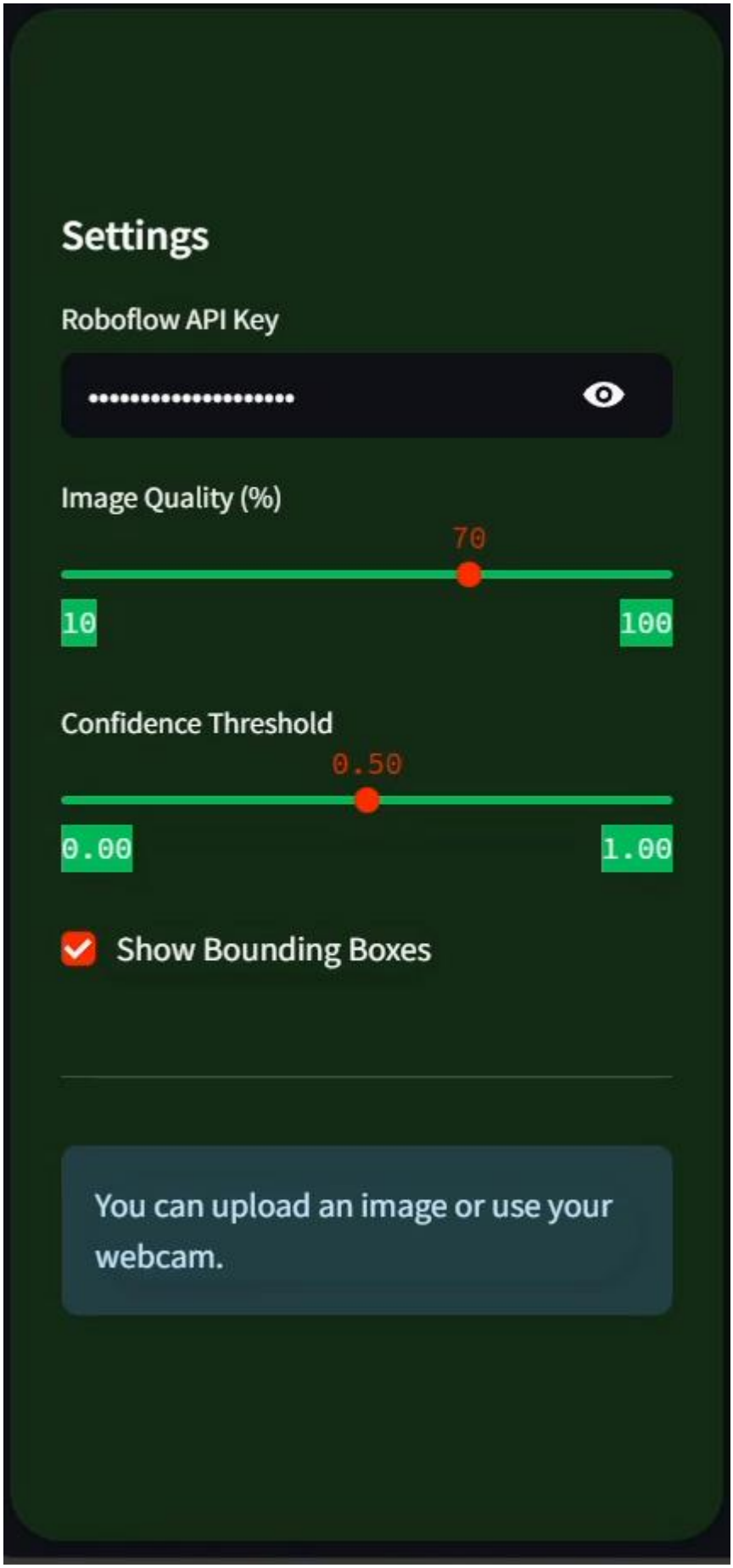
Input Image

Detected Plants

India-song (Confidence: 0.94)

BeeBeeTree (Confidence: 0.93)

Devil-Backbone (Confidence: 0.93)


Settings
Roboflow API Key
••••••••••••••••••••
Image Quality (%)
70
10
100
Confidence Threshold
0.50
0.00
1.00
Show Bounding Boxes
You can upload an image or use your webcam.

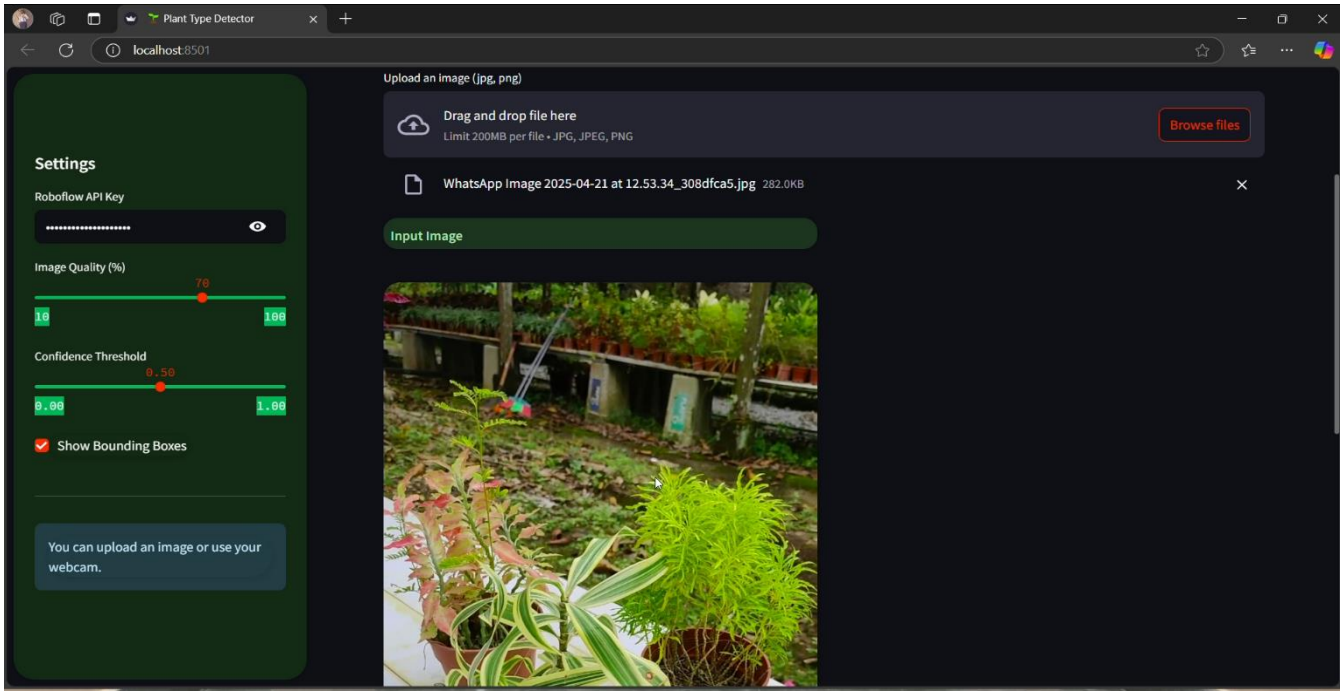
7.2 Upload/Capture Flow

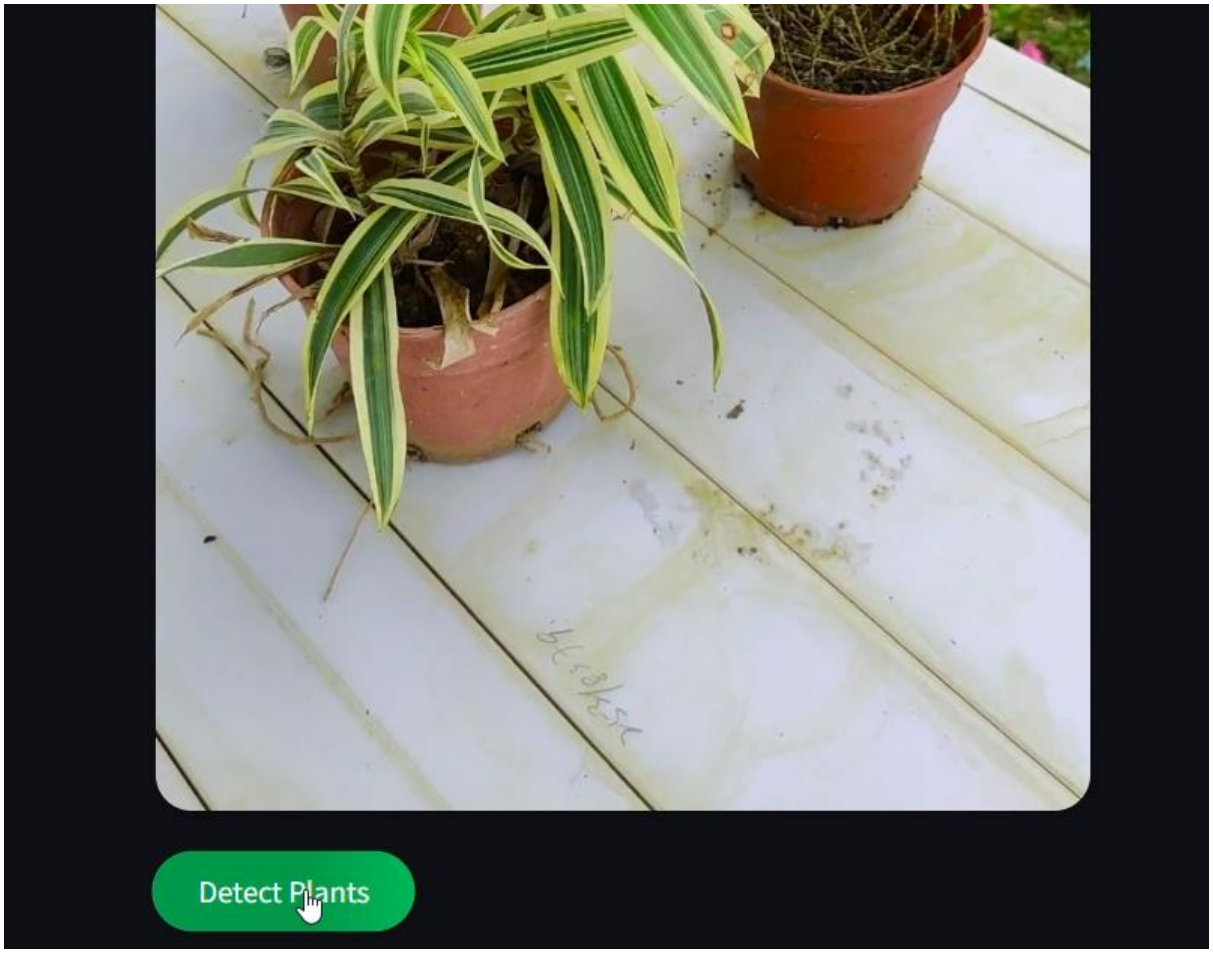
The UI supports two modes of image input:

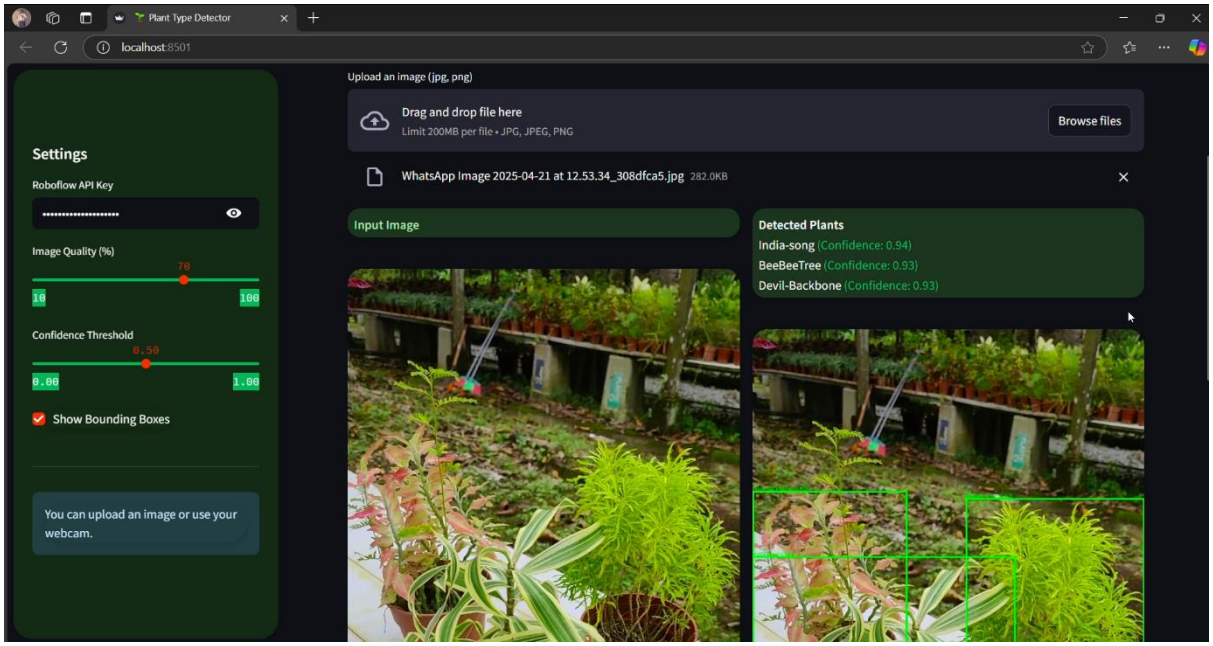
1. Upload Flow:

- User selects an image from their local storage using the upload widget.
- Upon clicking the “Detect” button, the image is passed through the detection pipeline.

- Results are shown below the upload area with boxes and labels overlayed.

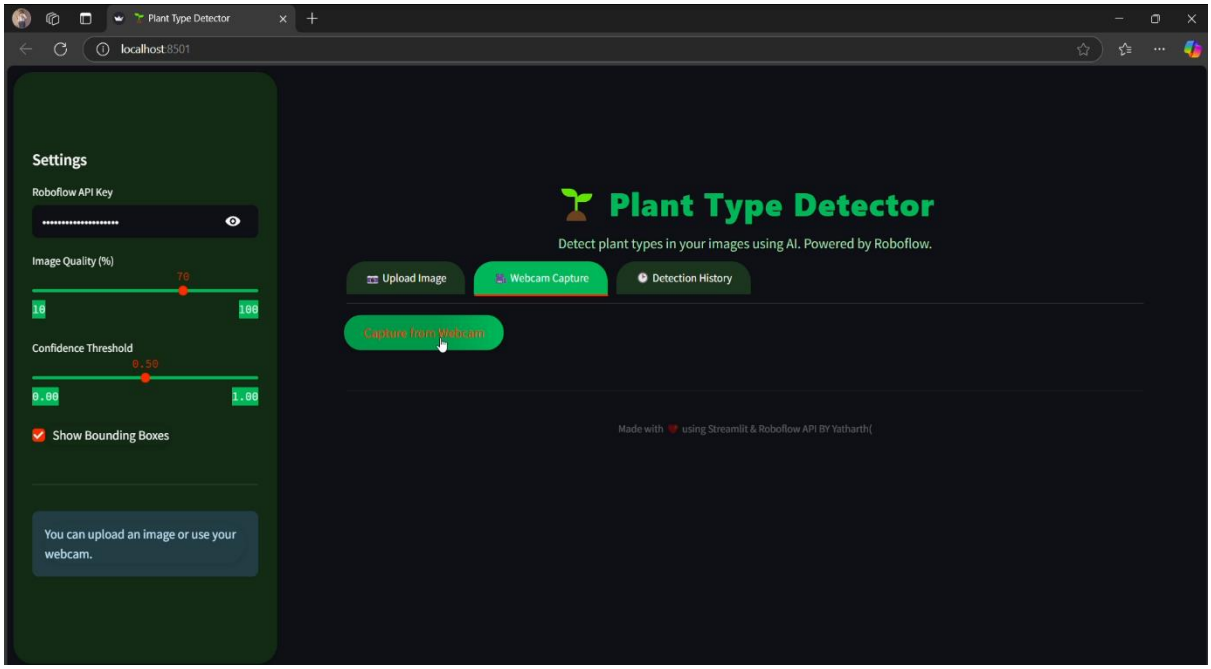


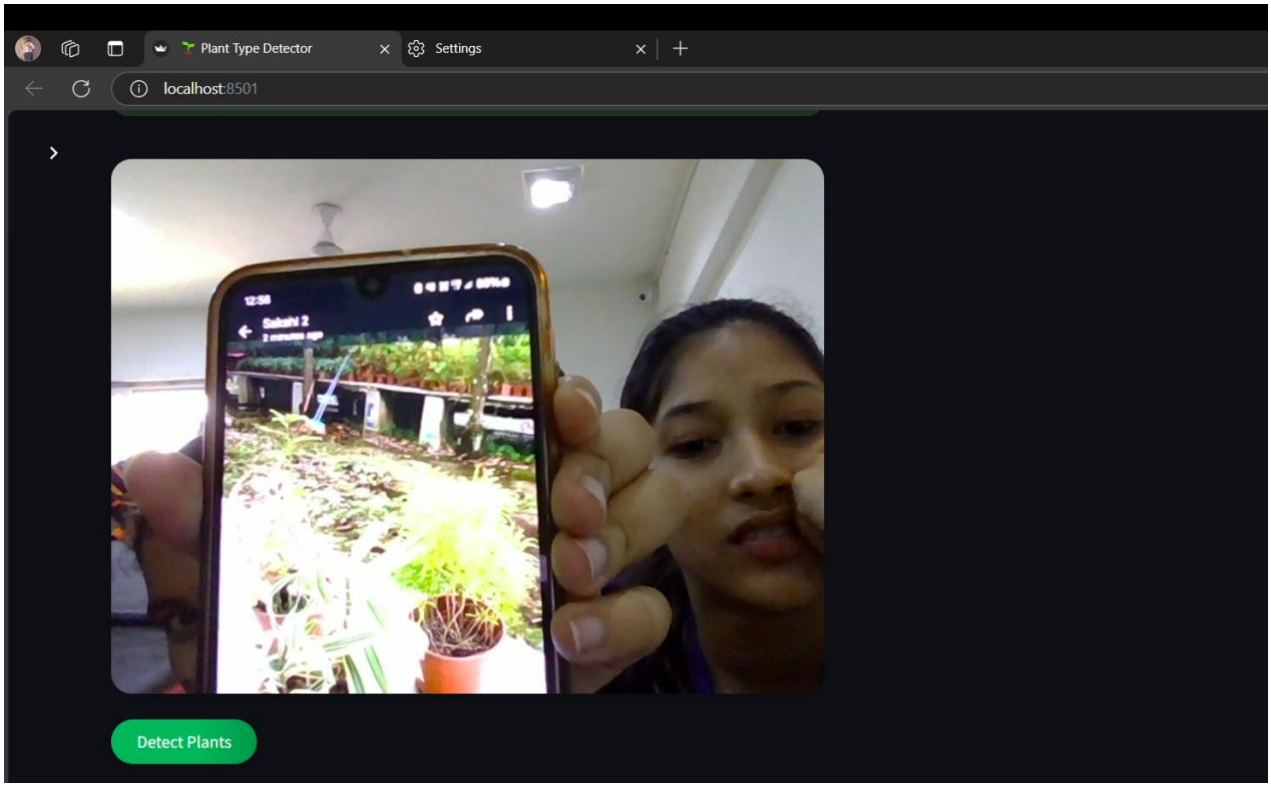

Detect Plants



2. Webcam Flow:

- The app accesses the device's webcam using OpenCV.
- The captured frame is frozen upon user command (button press).
- The frame is then treated as a static image and sent to the backend API.
- The processed image is shown with detections overlayed.





Chapter 8: AI/ML Integration

8.1 How Plant Detection Works

At the heart of the application lies an AI-based detection model capable of identifying plant species and parts from images. The model was sourced from Roboflow, a platform that offers a wide variety of trained machine learning models for vision tasks. The detection process leverages object detection principles, primarily using a convolutional neural network (CNN)-based architecture.

Key Concepts Behind Detection:

- **Object Detection:** The model scans the input image and detects multiple plant objects by drawing bounding boxes around them.
- **Classification:** Each detected object is classified into a plant category or species based on trained data.
- **Localization:** The coordinates of each object are calculated to enable visual display and bounding box placement.

The model is trained on thousands of annotated plant images. These annotations help the model learn patterns, textures, shapes, and colors commonly associated with different plant types.

8.2 Model Inference

Inference refers to the process where a trained model is used to make predictions on new data (unseen images). Here's how it works in this project:

Step-by-Step Inference Process:

1. **Image Encoding:**
    - The selected or captured image is converted to a base64 format (or appropriate binary format).
2. **API Communication:**
    - The encoded image is sent to Roboflow's cloud-hosted inference endpoint using a POST request. This is done through a REST API.
    - API credentials and model ID are securely passed to ensure authentication.
3. **Response Parsing:**
    - Roboflow returns a JSON object that includes details such as:
        - Detected class names
        - Confidence scores
        - Bounding box coordinates
4. **Visualization:**
    - The JSON data is used to draw boxes using OpenCV.
    - The results are then rendered on the front-end using Streamlit.

Advantages of Using Roboflow API:

- Quick integration without needing to train from scratch
- High accuracy due to use of pre-trained data
- Cloud processing eliminates need for local GPU

Chapter 9: Challenges Faced

9.1 Technical Challenges

1. API Integration Issues: Connecting the Roboflow API with the Streamlit app required proper authentication and formatting of image data. Initial attempts failed due to improper API key configuration and unsupported image types.

2. Image Format Compatibility: Not all uploaded or captured images were compatible with the inference model. Some images needed resizing or color format conversion before being processed.

3. Real-time Performance: Implementing webcam-based real-time detection with bounding boxes was challenging. Frame-by-frame processing slowed down the app when run on low-end hardware.

9.2 Development Challenges

1. Streamlit Layout Adjustments: Customizing the layout of the Streamlit app with dynamic image updates, titles, and sidebar options required experimentation. Some Streamlit widgets didn't work well when used together.

2. Error Handling: Errors from the API response (e.g., when no object was detected) had to be gracefully handled to avoid app crashes and provide meaningful messages to the user.

3. Time Management: Balancing academic work with this project was difficult. Completing the coding, testing, and documentation required proper scheduling and consistent effort.

9.3 Learning Curve

1. Roboflow API and Model Structure: Understanding how Roboflow handles inference and how to parse its JSON responses took time, especially for those new to REST APIs.

2. OpenCV for Visualization: Drawing bounding boxes accurately over an image with correct coordinates and labels was tricky, requiring precise calculations.

Despite these challenges, the project was successfully completed with robust performance and a user-friendly interface.

Chapter 10: Future Enhancements

10.1 Offline Mode One major limitation of the current implementation is the dependency on Roboflow's cloud-based API. A future upgrade could involve deploying a local inference engine using TensorFlow Lite or ONNX. This would enable offline functionality and real-time detection even in areas with limited or no internet access.

10.2 Plant Care Recommendations Adding a feature that not only detects the plant but also provides useful care tips—such as watering frequency, sunlight needs, soil type, and common diseases—would make the app more comprehensive and helpful for gardeners and farmers.

10.3 Support for More Plant Species The current model may be trained on a limited dataset. Future work can focus on expanding the dataset to cover a broader variety of plants, weeds, and crop species. Collaborations with agricultural research institutes could be beneficial for dataset collection.

10.4 Integration with Agriculture Platforms This app could be integrated with larger agricultural platforms or IoT devices to provide end-to-end solutions for crop monitoring, disease detection, and yield estimation.

10.5 User Account and History Introducing user authentication and a dashboard where users can save detection history and compare results over time can enhance engagement and repeat usage.

10.6 Multilingual Interface To make the app more inclusive, especially in rural or non-English-speaking regions, a multilingual interface can be developed to support regional languages.

10.7 Real-time Notifications and Alerts Using cloud services, push notifications can be implemented to alert users about potential threats or changes detected in plant health.

These enhancements would significantly improve the functionality, accessibility, and user satisfaction of the plant detection system.

Chapter 11: Conclusion

This project demonstrates the development of a user-friendly plant detection web application using Streamlit and AI-based inference via Roboflow. The integration of modern technologies like Python, OpenCV, and external inference APIs allows users to identify plant species through uploaded or real-time images with bounding boxes indicating predictions.

Through the implementation process, various core programming concepts and frameworks were explored, including image preprocessing, API communication, and real-time user interaction. The step-by-step methodology, from uploading images to receiving inference results, showcases the practical utility of combining machine learning with web-based platforms.

The project's significance lies in its relevance to agriculture and environmental monitoring. By enabling easy detection of plant species, the application can aid in plant identification for gardening, farming, and academic research. Though the current version is functional, the application has limitations such as reliance on an internet connection and a limited number of detectable species.

Looking ahead, the application has potential for meaningful upgrades such as offline capabilities, wider dataset integration, multilingual support, and plant care recommendations. These improvements would make the app more versatile and impactful in real-world scenarios.

Overall, the project was a valuable learning experience in software development, UI design, and AI integration, and it opens pathways for further research and development in the domain of AI-driven agriculture.

Chapter 12: Bibliography/References

- **Streamlit:** For creating the web application interface.
    - Streamlit. (2024). *Streamlit: The fastest way to build and share data apps*. https://streamlit.io/
- **PIL (Pillow):** For image processing.
    - Pillow (PIL). (2023). *Pillow 10.2.0 (PIL Fork)*. https://pypi.org/project/Pillow/
- **OpenCV (cv2):** For webcam integration and image manipulation.
    - Bradski, G., & Kaehler, A. (2008). *Learning OpenCV: Computer vision with the OpenCV library*. O'Reilly Media, Inc.
- **Roboflow Inference Server SDK:** For plant type detection using Roboflow's API.
    - Roboflow. (2024). *Inference SDK*. https://pypi.org/project/inference-sdk/
- **Other Libraries:**
    - Any other libraries used implicitly within the main libraries, such as numpy for numerical operations.