B.Tech. BCSE497J - Project-I

Rose Plant Disease Detection and Prevention

Bachelor of Technology

in

Computer Science and Engineering

by

M NRIPESH REDDY

Under the Supervision of

Mrs. Jeevana Jyothi Pujari

Assistant Professor Senior Grade 1

School of Computer Science and Engineering (SCOPE)

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

November 2024

DECLARATION

I hereby declare that the project entitled **Rose Plant Disease Detection and Prevention** submitted by me, for the award of the degree of *Bachelor of Technology in Computer Science and Engineering* to VIT is a record of bonafide work carried out by me under the supervision of **Mrs. Jeevana Jyothi Pujari.**

I further declare that the work reported in this project has not been submitted and will not be submitted, either in part or in full, for the award of any other degree ordiploma in this institute or any other institute or university.

Place : Vellore

Date : 20/11/2024

Signature of the Candidate

CERTIFICATE

This is to certify that the project entitled Rose Plant Disease Detection and Prevention submitted by Soma Uday Kiran(21BCE2959), M Nripesh Reddy(21BCE2424), V.P Praneeth Reddy (21BCT0353), **School of Computer Science and Engineering**, VIT, for the award of the degree of *Bachelor of Technology in Computer Science and Engineering*, is a record of bonafide work carried out by him / her under my supervision during Fall Semester 2024-2025, as per the VIT code of academic and research ethics.

The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this institute orany other institute or university. The project fulfills the requirements and regulations ofthe University and in my opinion meets the necessary standards for submission.

Place : Vellore

Date : 20/11/2024

Signature of the Guide

Examiner(s)

Umadevi K S

Bachelor of Technology

ACKNOWLEDGEMENTS

I am deeply grateful to the management of Vellore Institute of Technology (VIT) for providing me with the opportunity and resources to undertake this project. Their commitment to fostering a conducive learning environment has been instrumental in my academic journey. The support and infrastructure provided by VIT have enabled me to explore and develop my ideas to their fullest potential.

My sincere thanks to Dr. Ramesh Babu K, the Dean of the School of Computer Science and Engineering (SCOPE), for his unwavering support and encouragement. His leadership and vision have greatly inspired me to strive for excellence. The Dean's dedication to academic excellence and innovation has been a constant source of motivation for me. I appreciate his efforts in creating an environment that nurtures creativity and critical thinking.

I express my profound appreciation to Umadevi K S, the Head of the School of Computer Science and Engineering(SCOPE) for her insightful guidance and continuous support. Her expertise and advice have been crucial in shaping the direction of my project. The Head of Department's commitment to fostering a collaborative and supportive atmosphere has greatly enhanced my learning experience. Her constructive feedback and encouragement have been invaluable in overcoming challenges and achieving my project goals.

I am immensely thankful to my project supervisor, Mrs. Jeevana Jyothi Pujari for his dedicated mentorship and invaluable feedback. His patience, knowledge, and encouragement have been pivotal in the successful completion of this project. My supervisor's willingness to share his expertise and provide thoughtful guidance has been instrumental in refining my ideas and methodologies. His support has not only contributed to the success of this project but has also enriched my overall academic experience.

Thank you all for your contributions and support.

SOMA UDAY KIRAN

M NRIPESH REDDY

V.P PRANEETH REDDY

TABLE OF CONTENTS

List of Figures

List of Tables

List of Abbreviations

AI	Artificial Intelligence
API	Application programming interface
IoT	Internet of Things
ML	Machine Learning
CNN	Convolutional Neural Network
GUI	Graphical user interface
GLCM	Gray-Level Co-Occurrence Matrix

ABSTRACT

Nowadays plants are suffering many diseases due to widespread use of pesticides and sprays but identifying rotten areas of plants in the early stage can save plants. Examination of plants disease literally means examining various observable patterns on plants. Manually detecting disease in plants can be a tiresome process, hence image processing can do wonders in this context. Plant disease can be seen in different parts like in stem, root, shoot and even in fruit.

Detection of plant disease by the automatic way not only reduces time but also it is able to save the plant from the disease in the beginning stage itself. We use different image processing techniques to predict the problem in plants. We basically deal with the rose plants and flowers, we will detect the various kinds of diseases in the rose plants and flowers we will highlight the affected part and classify according to the disease datasets .we are using 4 different datasets, 3 datasets are disease datasets and 1 dataset is healthy plant dataset. Then we will provide the solution for the detected disease which will help the farmer in good cultivation and the more profits. This project focuses on the early detection of rose plant diseases using deep learning techniques, particularly Separable Convolutional Neural Networks (CNNs) and K-means clustering. The system is designed to identify diseases like Black Spots, Powdery Mildew, Downy Mildew, and Botrytis Blight by analyzing images of rose leaves and flowers. It leverages two datasets—one for training and the other for testing—to ensure model accuracy.

1. INTRODUCTION

1.1 Background

Rose plants, cherished for their beauty and economic value, are prone to various diseases such as Black Spots, Powdery Mildew, Downy Mildew, and Botrytis Blight. These diseases can severely impact plant health, leading to decreased yields and quality, and resulting in significant financial losses for growers. Traditional methods of detecting plant diseases often rely on manual inspection, which is labor-intensive and time-consuming. To address these challenges, this project aims to develop an automated system for early disease detection using advanced image processing and machine learning techniques. By analyzing images of rose plants, the system identifies and classifies disease symptoms efficiently. The integration of depthwise separable convolutions and transfer learning enhances the model's accuracy while reducing computational requirements. This approach not only improves disease management and resource allocation but also supports sustainable agricultural practices by minimizing pesticide use and labor costs.

1.2 Motivation

The motivation behind this project is driven by the critical need for efficient rose plant disease management due to its significant economic and environmental implications. Rose cultivation, a lucrative segment of agriculture, faces severe financial losses from diseases like Black Spots, Powdery Mildew, Downy Mildew, and Botrytis Blight. Traditional manual inspection methods are not only labor-intensive but also prone to errors and inefficiencies. This project seeks to address these issues by developing an automated disease detection system using advanced image processing and machine learning. By leveraging techniques such as depthwise separable convolutions and transfer learning, the system aims to enhance detection accuracy and reduce computational demands. This innovation promises to streamline disease management, reduce the reliance on pesticides, and lower labor costs, thereby supporting more sustainable and economically viable farming practices. Ultimately, the project aims to improve plant health and boost profitability for rose growers.

1.3 Scope of the Project

The scope of this project involves developing an automated system for early detection and classification of rose plant diseases. It includes identifying four specific diseases—Black Spots, Powdery Mildew, Downy Mildew, and Botrytis Blight—by analyzing images of rose leaves, flowers, and stems. The project integrates advanced image processing techniques, such as grayscale conversion, K-means clustering for segmentation, and the Gray-Level Co-Occurrence Matrix (GLCM) for texture analysis, to enhance disease detection. A machine learning model, employing depthwise separable convolutions and transfer learning, will be trained to accurately classify these diseases while optimizing computational efficiency. An intuitive graphical user interface (GUI) will be developed to enable users to upload images, receive disease predictions, and obtain treatment recommendations. The system aims to support sustainable agricultural practices by reducing pesticide use and labor costs, and will be tested both in controlled environments and real-world conditions to ensure its effectiveness and reliability.

2. PROJECT DESCRIPTION AND GOALS

2.1 Literature Review

[1] Proceedings of the 7th International Conference on Trends in Electronics and Informatics (ICOEI 2023) IEEE Xplore the author Ali- Al - Alvy, Md. Published an article on Rose Plant Disease Detection using Deep Learning. This article presents an extensive dataset of rose leaves images, both diseases affected and diseases are classified into three classes (Blackspot, Downy Mildew, and Fresh Leaf). The dataset is composed of the collected images which were captured during the seasonal time of disease affection with the consultation of a domain expert and the dataset is accessible.

[2] IEEE International Conference on Robotics, Automation, Artificial-Intelligence and Internet-of-Things (RAAICON) the author Sahadat Hossain Khan published a paper on
Rose Leaf Disease Detection Using CNN provides an algorithm for best image processing and this processing help to find out the best solution. They used a large type of image-based data set for the accuracy of the image solution.

[3] International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT) (2020) the author Thoma sarkar mahbubur rehman published a paper on Rose diseases recognition using mobilenet. In this paper, they have used transfer learning and without transfer learning technique by using a MobileNet model to detect rose diseases. Augmentation has been performed on the collected image data for the lack of many images. For experimental purpose, 1600 data images are used to train the model and 400 data images are used to test the model. Within two approaches, MobileNet with transfer learning omits the MobileNet without transfer learning technique and achieves 95.63% accuracy. The acquired result exhibits that the working method for recognizing rose diseases is appeasement and feasible.

[4] Creative commons attribution international (2020) the author Paramashivam, Vijaya lakshmi published a paper on Intelligent plant disease identification using machine learning. In this work, a real-time decision support system integrated with a camera sensor module was designed and developed for identification of plant disease. Furthermore, the performance of three machine learning algorithms, such as Extreme Learning Machine (ELM) and Support Vector Machine (SVM) with linear and polynomial kernels was analyzed. Results demonstrate that the performance of the extreme learning machine is better when compared to the adopted support vector machine classifier. It is also observed that the sensitivity of the support vector machine with

a polynomial kernel is better when compared to the other classifiers.

[5] IEEE Eurasia Conference on IOT, Communication and Engineering (ECICE) (2019) the author Sammy Miltane, Bobby dioquino published a paper on Plant Leaf Detection and Disease Recognition using Deep Learning. This study provides an efficient solution for detecting multiple diseases in several plant varieties. The system was designed to detect and recognize several plant varieties, specifically apple, corn, grapes, potato, sugarcane, and tomato. The system can also detect several diseases of plants. The trained model has achieved an accuracy rate of 96.5% and the system was able to register up to 100% accuracy in detecting and recognizing the plant variety and the type of diseases the plant was infected.

2.2 Research Gap

The timely identification and early prevention of crop diseases are essential for improving production. In this paper, deep convolutional-neural-network (CNN) models are implemented to identify and diagnose diseases in plants from their leaves, since CNNs have achieved impressive results in the field of machine vision. Standard CNN models require a large number of parameters and higher computation cost. In this paper, we replaced standard convolution with depth separable convolution, which reduces the parameter number and computation cost. The implemented models were trained with an open dataset consisting of 14 different plant species, and 38 different categorical disease classes and healthy plant leaves. To evaluate the performance of the models, different parameters such as batch size, dropout, and different numbers of epochs were incorporated. The implemented models achieved a disease classification accuracy rates of 98.42%, 99.11%, 97.02%, and 99.56% using InceptionV3, InceptionResNetV2, and EfficientNetB0, respectively, which were greater than that of traditional handcrafted-feature-based approaches. In comparison with other deep-learning models, the implemented model achieved better performance in terms of accuracy and it required less training time. Moreover, the MobileNetV2 architecture is compatible with mobile devices using the optimized parameter. The accuracy results in the identification of diseases showed that the deep CNN model is promising and can greatly impact the efficient identification of the diseases, and may have potential in the detection of diseases in real-time agricultural systems.

2.3 Objectives

The primary objective of this project is to develop an automated system for early detection and classification of diseases in rose plants. The project aims to achieve the following SMART goals:

Disease Detection: Develop and implement a system capable of identifying and classifying four specific rose plant diseases—Black Spots, Powdery Mildew, Downy Mildew, and Botrytis Blight—with a classification accuracy of at least 95% by the end of the project.

Image Processing: Integrate advanced image preprocessing techniques, including grayscale conversion, K-means clustering for segmentation, and Gray-Level Co-Occurrence Matrix (GLCM) for texture analysis, to enhance disease detection within a timeline of 3 months.

Machine Learning Model: Train and optimize a machine learning model using depthwise separable convolutions and transfer learning to achieve high classification accuracy while maintaining computational efficiency, with model training completed within 4 months.

User Interface Development: Design and deploy an intuitive graphical user interface (GUI) that allows users to upload images, receive disease predictions, and access treatment recommendations, with the GUI development and deployment achieved within 5 months.

Sustainability Impact: Reduce the reliance on pesticides and labor costs by providing accurate and timely disease detection, aiming to demonstrate a reduction in pesticide use by 20% and a decrease in labor costs by 15% within 6 months of deployment.

Validation: Conduct comprehensive testing of the system in both controlled lab conditions and real-world field environments to ensure reliability and effectiveness, with validation and final adjustments completed within 6 months.

2.4 Problem Statement

The identification of the rose plant diseases is a key for preventing the losses in the yield and quality of agriculture products. Diseases decrease the productivity of the plant and it also restricts the growth of the plant, and both quality and quantity of plant gets reduced. Disease detection on plants is very critical for sustainable agriculture. It is very hard to monitor plant diseases with the hands. It needs a very great amount of work, expert knowledge in plant diseases, and also needs more than enough processing time.

2.5 Project Plan

ID	Task Name	Start	End	Duration
1	TITLE OF THE PROJECT	2024-07-15	2024-07-19	5 days
2	ABSTRACT	2024-07-22	2024-07-24	3 days
3	INTRODUCTION	2024-07-25	2024-07-26	2 days
4	LITERATURE SURVEY	2024-07-29	2024-07-31	3 days
5	COLLECTION OF DATASET	2024-08-01	2024-08-02	2 days
6	DATA PREPROCESSNG	2024-08-05	2024-08-15	9 days
7	FEATURE EXTRACTION	2024-08-16	2024-09-02	12 days
8	MODEL TRAINING	2024-09-03	2024-09-18	12 days
9	HYBRID MODEL CREATION	2024-09-19	2024-10-01	9 days
10	MODEL EVALUATION	2024-10-02	2024-10-04	3 days
11	CONDUCT PROJECT REVIEW	2024-10-07	2024-10-08	2 days
12	HANDOVER DELIVERABLES	2024-10-09	2024-10-10	2 days

Figure 1 : Project Duration

Above table outlines a project timeline, starting from July 15, 2024 . The table provides a clear and concise overview of the project's phases, their start and end dates, and their estimated durations.

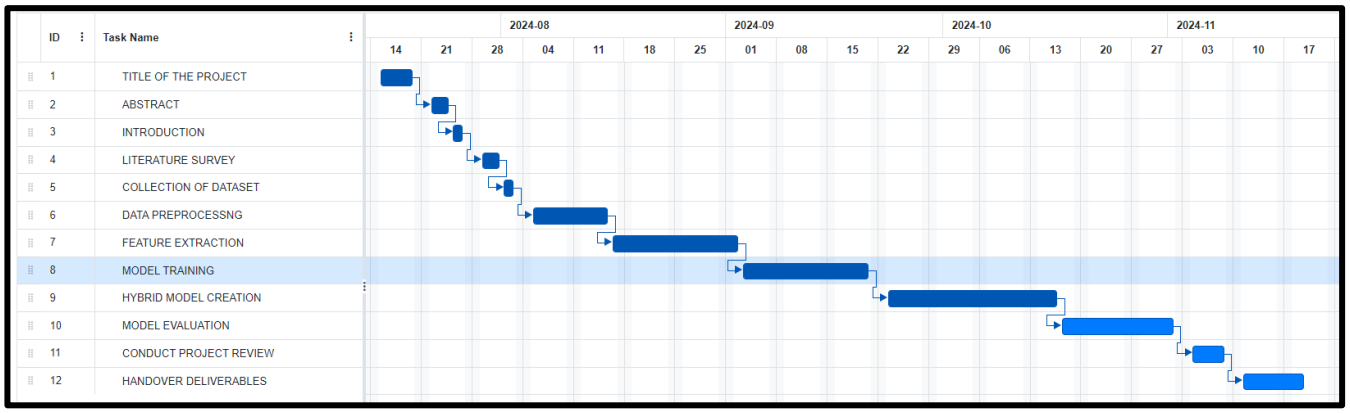

Figure 2 : Gantt Chart

The Gantt chart provides a visual representation of the project's schedule, allowing for easy tracking of task progress and potential bottlenecks.

3. TECHNICAL SPECIFICATION

3.1 Requirements

For the Rose Plant Disease Detection system, both functional and non-functional requirements need to be addressed to ensure the project's success:

3.1.1 Functional

Functional Requirements outline the essential functionalities that the system must provide to meet the objectives of the project. Below are the key functional requirements for the rose plant disease detection system:

1. **Image Upload and Input Handling**:
    - The system should allow users to upload images of rose plant leaves or flowers in supported formats (e.g., JPEG, PNG).
    - It should validate image quality, ensuring that the uploaded images meet resolution and clarity requirements for accurate analysis.
2. **Image Preprocessing**:
    - The system must perform preprocessing on the uploaded images, including grayscale conversion and noise reduction.
    - K-means clustering should be used to segment the images for extracting meaningful features relevant to disease identification.
3. **Disease Detection and Classification**:
    - The core functionality is to analyze the input image and accurately classify one of the four rose plant diseases: Black Spots, Powdery Mildew, Downy Mildew, or Botrytis Blight.
    - The classification should be performed using a trained Convolutional Neural Network (CNN) model with depthwise separable convolutions and transfer learning.
4. **Result Display and Feedback**:
    - After analysis, the system should display the disease prediction along with the confidence level.

- It should provide users with treatment recommendations and preventive measures based on the diagnosed disease.

5. **Model Training and Optimization**:
    - The system should be capable of training and updating the machine learning model with new data to improve accuracy over time.
    - It should support re-training with new datasets to enhance detection of additional diseases if required.
6. **User Interface (UI)**:
    - The system must feature a user-friendly UI where users can easily upload images and receive results.
    - It should include clear navigation, visual aids, and provide detailed instructions for users who are not familiar with the technology.
7. **Performance Monitoring**:
    - The system should log and report performance metrics, including processing time, accuracy, and any errors encountered.
    - It should provide diagnostic reports for further model evaluation and improvements.
8. **Integration with External Systems**:
    - The system should have the capability to integrate with external agricultural databases or APIs for updated disease information, treatment methods, and image datasets.

3.1.2 Non-Functional

Non-Functional Requirements focus on the system's quality attributes and operational capabilities. They ensure that the system performs efficiently, reliably, and securely. Below are the key non-functional requirements for the rose plant disease detection system:

Performance:

The system should process and classify images within 2-5 seconds, ensuring timely feedback to users. It must be capable of handling at least 100 simultaneous users without a noticeable drop in performance.

Scalability:

The system should be designed to scale as the number of users increases, both vertically (more powerful hardware) and horizontally (additional servers).It must accommodate growth in dataset size, allowing for new diseases or features to be incorporated over time.

Reliability:

The system should have 99.9% uptime to ensure continuous availability for users.In the case of a failure, there should be a robust failover mechanism that allows for quick recovery.

Accuracy:

The disease detection model should aim for an accuracy rate of at least 90%, minimizing false positives and false negatives in disease diagnosis.The system should be regularly tested and updated to maintain high precision and recall.

Security:

User data, including uploaded images, should be securely transmitted using HTTPS and encrypted when stored.Access control mechanisms should prevent unauthorized access to the system, ensuring that only authenticated users can upload and retrieve data.

Usability:

The user interface should be intuitive and easy to navigate, even for individuals without a technical background.Clear instructions and visual indicators should guide users through the image upload and disease diagnosis process.

Maintainability:

The codebase should be modular and well-documented to facilitate easy updates, bug fixes, and new feature integration.Regular maintenance checks should be performed to ensure optimal system performance.

Portability:

The system should be platform-independent, allowing it to run on various operating systems, including Windows, macOS, and Linux.It should also be deployable on cloud infrastructure for wider accessibility.

Data Privacy:

User images and results should be anonymized, with data retention policies ensuring that personal data is stored only as long as necessary.The system should comply with data privacy regulations such as GDPR or local equivalents.

Extensibility:

The system should be designed in a way that allows future enhancements, such as adding new plant species or expanding disease detection capabilities without significant architectural changes.

These non-functional requirements ensure the system operates efficiently and provides a smooth, secure, and scalable user experience.

3.2 Feasibility Study

Feasibility Study is a critical evaluation process that assesses the practicality of a project or system. It helps determine whether the project is viable, identifying potential challenges, constraints, and opportunities. The study typically includes four key aspects

3.2.1 Technical Feasibility

Technical Feasibility assesses whether the technology required for the project is available, reliable, and capable of meeting the project's objectives. For the rose plant disease detection system, this analysis examines the following key areas:

Image Processing and Machine Learning Algorithms: The project relies on advanced image processing techniques and machine learning models, such as Convolutional Neural Networks (CNNs) with depthwise separable convolutions and transfer learning. These are established technologies in the field of computer vision and are suitable for classifying diseases from plant images. The availability of frameworks like TensorFlow, Keras, and OpenCV ensures that these algorithms can be implemented efficiently.

Hardware Requirements: The system requires access to high-quality cameras for capturing clear images of rose plants, as well as computing resources to process the images and train the machine learning models. Cloud computing platforms (e.g., AWS, Google Cloud) offer scalable processing power for large datasets, making this aspect technically feasible.

Software Development Tools: Python, along with popular libraries for machine learning and image processing, provides a robust environment for developing and deploying the system. These tools are well-documented and widely used, minimizing technical risks.

Scalability and Performance: Depthwise separable convolutions reduce the computational load, making the system more efficient and scalable for real-time disease detection. The integration of transfer learning further ensures the model can be trained on smaller datasets while achieving high accuracy.

Overall, the technical resources and tools required for this project are readily available, making the system technically feasible for implementation.

3.2.2 Economic Feasibility

Economic Feasibility evaluates whether the project is financially viable, assessing if the benefits outweigh the costs involved. For the rose plant disease detection system, the economic feasibility can be analyzed as follows:

Initial Development Costs: The main expenses will include setting up the image processing system, developing machine learning models, acquiring datasets, and investing in hardware such as cameras and computing resources. If cloud-based services are used for computation (e.g., AWS or Google Cloud), subscription costs should also be considered. However, these expenses can be minimized by utilizing open-source tools like TensorFlow, Keras, and OpenCV.

Operational Costs: After the system is developed, the ongoing costs include maintaining the system, updating the model as new disease data becomes available, and managing the hardware. These costs are relatively low compared to the potential benefits.

Benefits: The system significantly reduces labor costs associated with manual disease inspection, increases the accuracy of early disease detection, and minimizes the use of pesticides by targeting only infected plants. This leads to better resource allocation and higher crop yields, ultimately improving the profitability for rose growers.

Return on Investment (ROI): By automating disease detection and improving crop health, growers can expect a reduction in losses due to disease and an increase in productivity. Over time, the cost savings and improved yield quality will justify the initial investment, making the project economically viable.

In conclusion, the long-term financial benefits, combined with lower operational costs, indicate that the project is economically feasible.

3.2.3 Social Feasibility

Social Feasibility assesses how well a project will be accepted by the people it affects and its broader societal impact. For the rose plant disease detection system, social feasibility can be examined in the following ways:

1. Acceptance by Growers: The automated system offers a significant advantage to rose plant growers by simplifying disease detection, reducing the time and labor required for manual inspections. Growers who adopt this technology can expect improved efficiency and crop health.

2. Impact on Employment: Although automation may reduce the need for manual labor in disease detection, it can create new opportunities in tech-driven agricultural roles, such as system operators and data analysts. The system could also support smaller-scale farmers by providing them with affordable, efficient disease detection tools, helping them compete with larger operations.

3. Environmental and Health Benefits: By promoting early and accurate disease detection, the system encourages precise pesticide use, reducing unnecessary chemical applications. This not only supports sustainable farming practices but also protects the environment and public health by limiting chemical runoff and exposure.

4. Community Impact: The adoption of technology-driven agricultural practices can help build more resilient farming communities, improve overall productivity, and potentially lead to higher incomes for farmers. The positive economic effects can extend to related industries, such as floristry and horticulture.

In summary, the project is socially feasible, as it offers clear benefits in terms of efficiency, sustainability, and community growth, though it may require educational outreach to ensure widespread adoption.

3.2 System Specification

3.2.1 Hardware Specification

- Windows7/8/10 or Mac OS X 10.11 or higher, 64-bit or Linux.
- Memory-10 GB minimum.
- Ram-4 to 8GB.

3.2.2 Software Specification

Programming Language:

- Python 3.7 or later, due to its extensive library support for data science and machine learning.

Development Tools and Libraries:

- Libraries for data preprocessing and analysis: pandas, NumPy, scikit-learn.

- Machine learning frameworks: K-Means, TensorFlow.

- Visualization tools: Matplotlib, Seaborn for generating insights.

Integrated Development Environment (IDE):

- PyCharm, Jupyter Notebook, or VS Code for streamlined development.

Version Control:

- Git/GitHub for tracking code changes and collaboration.

Security Software:

- Tools for secure data handling and encryption (e.g., OpenSSL).

The outlined feasibility studies and specifications confirm that the project is technically, economically, and socially viable. With the proper infrastructure and resources, the malware detection system promises to deliver robust and efficient cybersecurity solutions.

4.DESIGN APPROACH AND DETAILS

4.1 SYSTEM ARCHITECTURE:

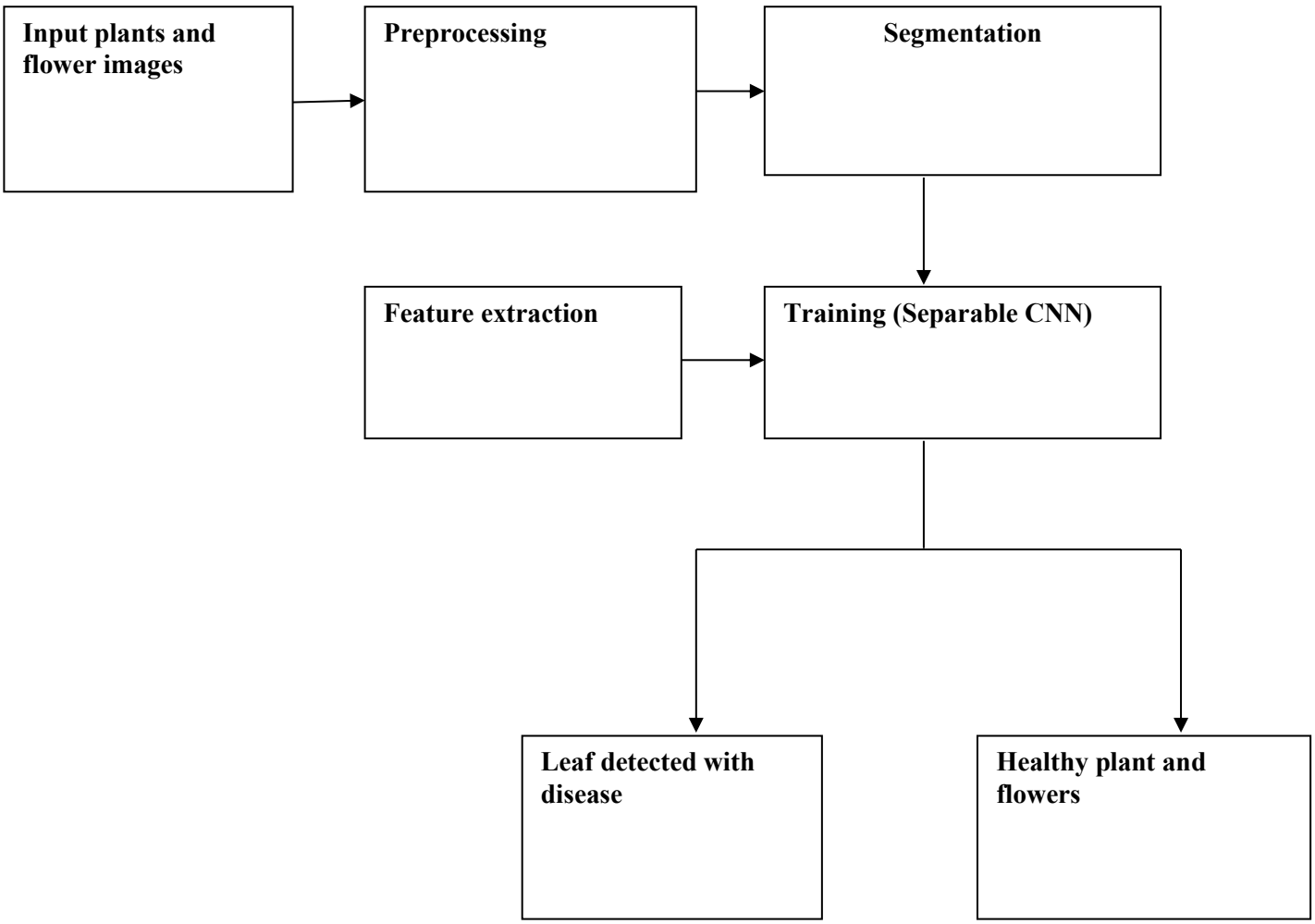

Figure 3 : System Architecture

System Architecture Overview:

The **Rose Plant Disease Detection and Prevention** system employs a multi-layered architecture to efficiently detect and manage plant diseases. At the top, the **User Interface (UI)** allows users, such as farmers or gardeners, to upload images of rose plants and view disease predictions and treatment recommendations. The system processes these images through various stages, starting with the **Preprocessing Layer**, where the images are converted to grayscale to simplify analysis by reducing color complexity. Next, the **Image Processing Layer** uses K-means clustering to segment the image, isolating areas of interest, such as diseased spots, from healthy parts of the plant. In the **Feature Extraction Layer**, important features like texture patterns are extracted using techniques like the Gray-Level Co-occurrence Matrix (GLCM) to aid in disease classification.

The core of the system lies in the **Machine Learning and Prediction Layer**, where a Separable Convolutional Neural Network (CNN) is trained to classify diseases based on the extracted features. This deep learning model is trained on a dataset of rose plant images to identify diseases such as Black Spots, Powdery Mildew, Downy Mildew, and Botrytis Blight. Once the disease is identified, the **Recommendation and Prevention Layer** provides treatment recommendations specific to the predicted disease, which are displayed to the user.

All interactions, predictions, and feedback are managed and stored in the **Database Layer**, ensuring that the system can track predictions and treatment history for continuous improvement. The system's architecture integrates various image processing techniques, machine learning models, and databases to offer an efficient and scalable solution for early disease detection and management. This approach helps users take preventive actions in a timely manner, promoting healthier rose plant cultivation and improving agricultural practices.

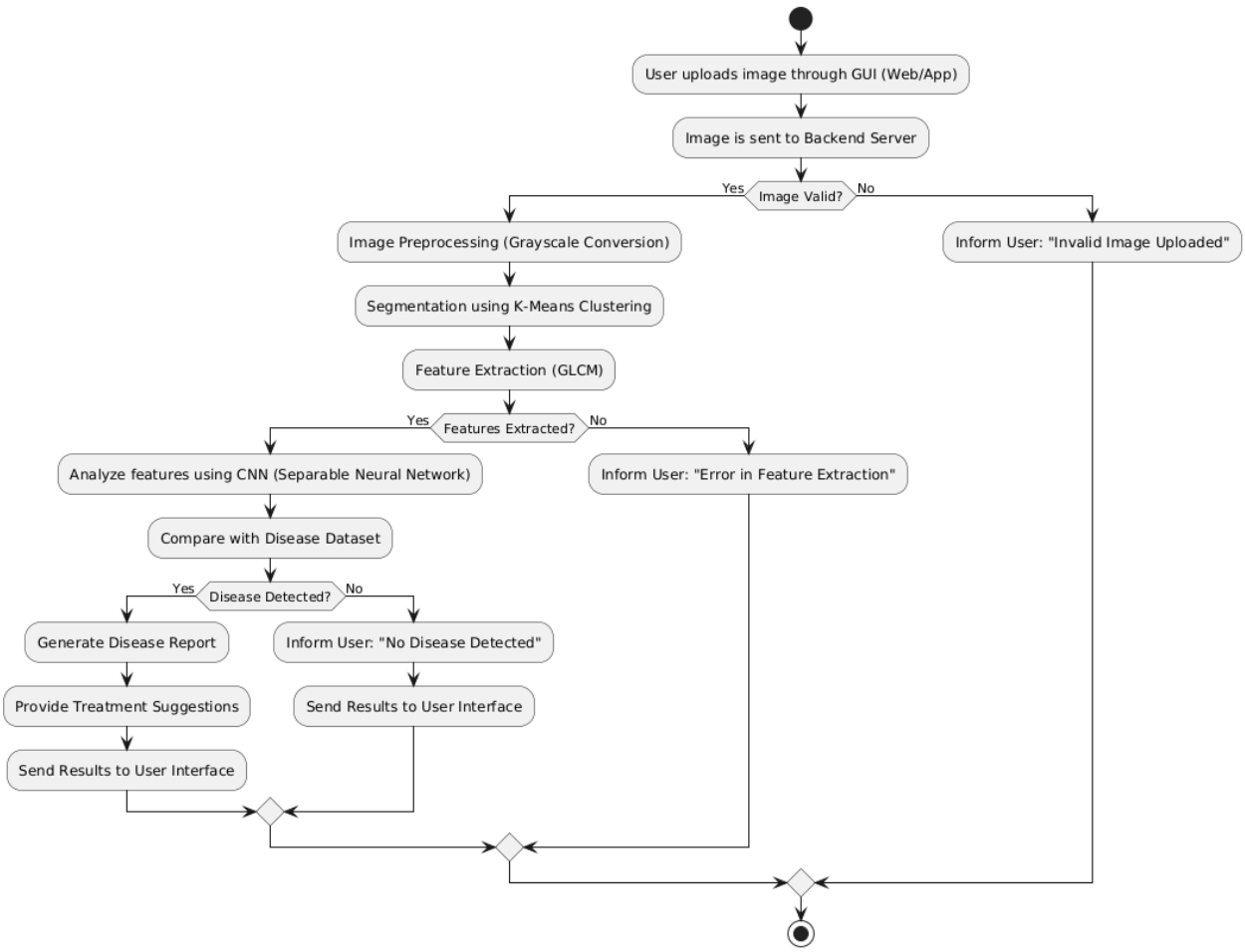

Figure 4 : Flow Chart of Architecture

4.2 DESIGN :

4.2.1 Data Flow Diagram:

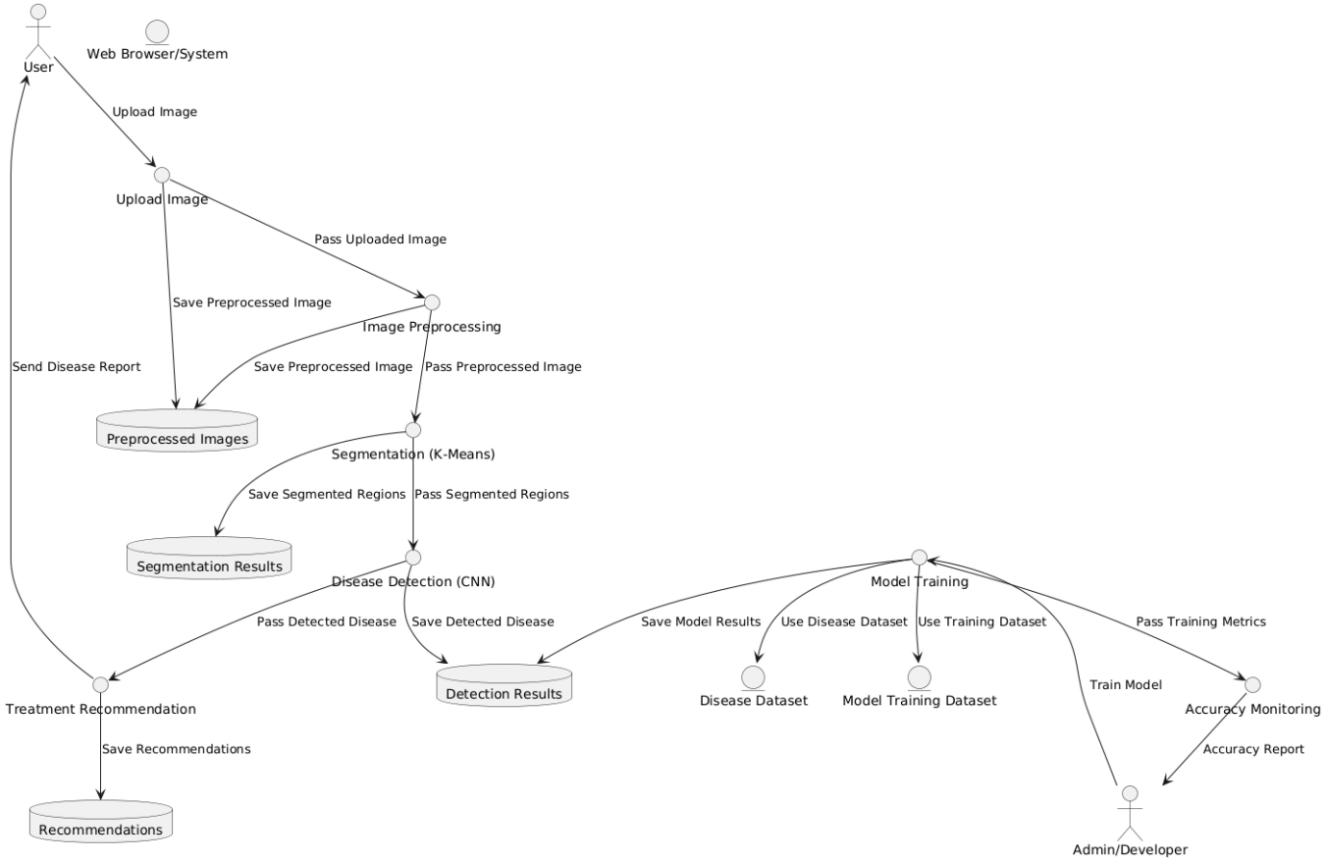

Figure 5 : Data Flow Diagram

Components and Data Flow:

1. **External Entity (User)**
    - Provides input image of a rose plant
    - Receives disease prediction and treatment recommendations
2. **Image Preprocessing**
    - Transforms input image through multiple stages:
        - Image loading
        - Grayscale conversion
        - Noise reduction
        - Image resizing

        - Segmentation (K-means clustering)
        - Feature extraction (GLCM)
    - Prepares image for further analysis
3. **Model Training**
    - Processes training data through:
        - Data splitting (training/testing sets)
        - Defining model architecture
        - Training the model
        - Evaluating model performance
    - Creates a trained predictive model
4. **Disease Prediction**
    - Uses trained model to:
        - Perform model inference
        - Generate prediction output
    - Classifies rose plant disease from preprocessed image
5. **Image Database**
    - Stores preprocessed images
    - Maintains labeled training data
    - Supports model training and future references

Key Interactions:

- User image flows through preprocessing
- Preprocessed image feeds into model training and prediction
- Trained model enables disease classification
- Prediction results return to user

4.2.2 Use Case Diagram:

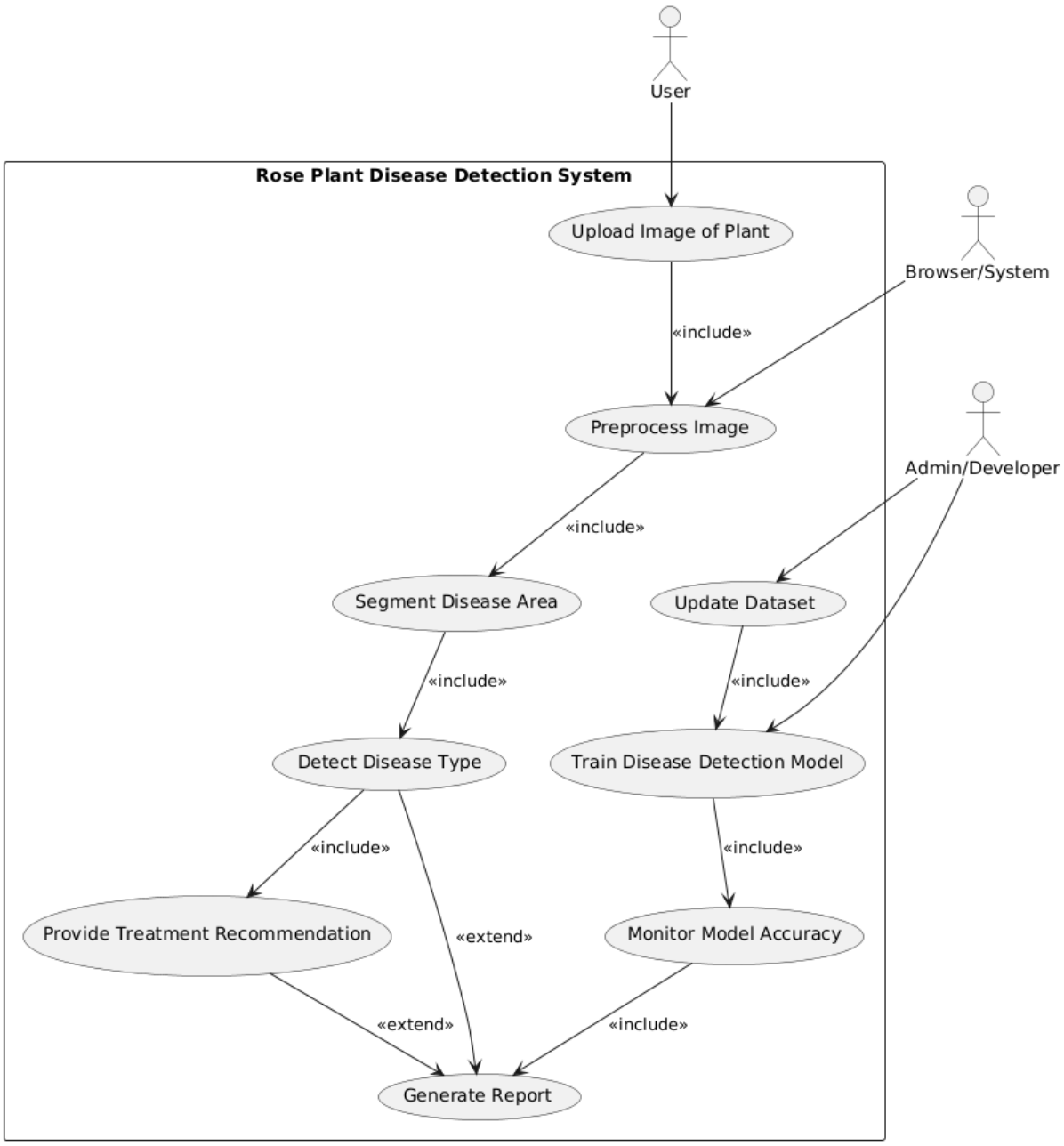

Figure 6 : Use Case Diagram

Use Case Diagram Description

Actors:

1. **User:**
    - Primary actor who interacts with the system to upload images and receive results.
    - Includes farmers, gardeners, or agricultural experts seeking to diagnose rose plant diseases.

2. **System:**
    - The backend of the application responsible for processing images, analyzing diseases, and generating results.
3. **External Database:**
    - Stores pre-trained disease models and datasets for feature matching and classification.

Use Cases:

1. **Upload Image:**
    - The user uploads an image of the rose plant (leaves, flowers, stems, or roots) through the system's GUI.
    - Triggers the process for disease detection.
2. **Validate Image:**
    - The system checks if the uploaded image is in a valid format and resolution.
    - Invalid images prompt an error message to the user.
3. **Preprocess Image:**
    - Converts the image to grayscale to simplify analysis.
    - Segments the image using K-means clustering to isolate diseased areas.
4. **Extract Features:**
    - The system uses Gray-Level Co-Occurrence Matrix (GLCM) to extract texture features of the image.
    - Captures details like spots, mildew patches, or other abnormalities.
5. **Classify Disease:**
    - A Convolutional Neural Network (CNN) analyzes the extracted features and matches them against the pre-trained dataset.
    - Diseases such as Black Spots, Powdery Mildew, Downy Mildew, and Botrytis Blight are identified.

6. **Generate Results:**
    - The system generates a detailed report that includes:
        - Identified disease (if any).
        - Recommended treatments or preventive measures.
7. **Provide Treatment Suggestions:**
    - The system retrieves treatment options for the detected disease from the database.
    - Results are sent back to the user interface.
8. **Save Results:**
    - Allows users to download or save the report for future reference.

Relationships:

1. **Include:**
    - Preprocess Image is included in the disease classification process since preprocessing is a mandatory step.
    - Generate Results always includes disease classification to provide meaningful output.
    - Provide Treatment Suggestions is included when a disease is detected.
2. **Extend:**
    - Save Results extends the use case to optionally store or export the analysis report.
    - Validate Image is extended to handle invalid uploads.

Description with Example:

- Scenario: A farmer notices unusual spots on rose plant leaves. Using this system:
    1. The farmer uploads an image via the GUI.
    2. The system validates the image and preprocesses it.
    3. The CNN model classifies the disease as Black Spots.
    4. The system generates a report recommending specific fungicides and preventive measures.

4.2.3 Class Diagram:

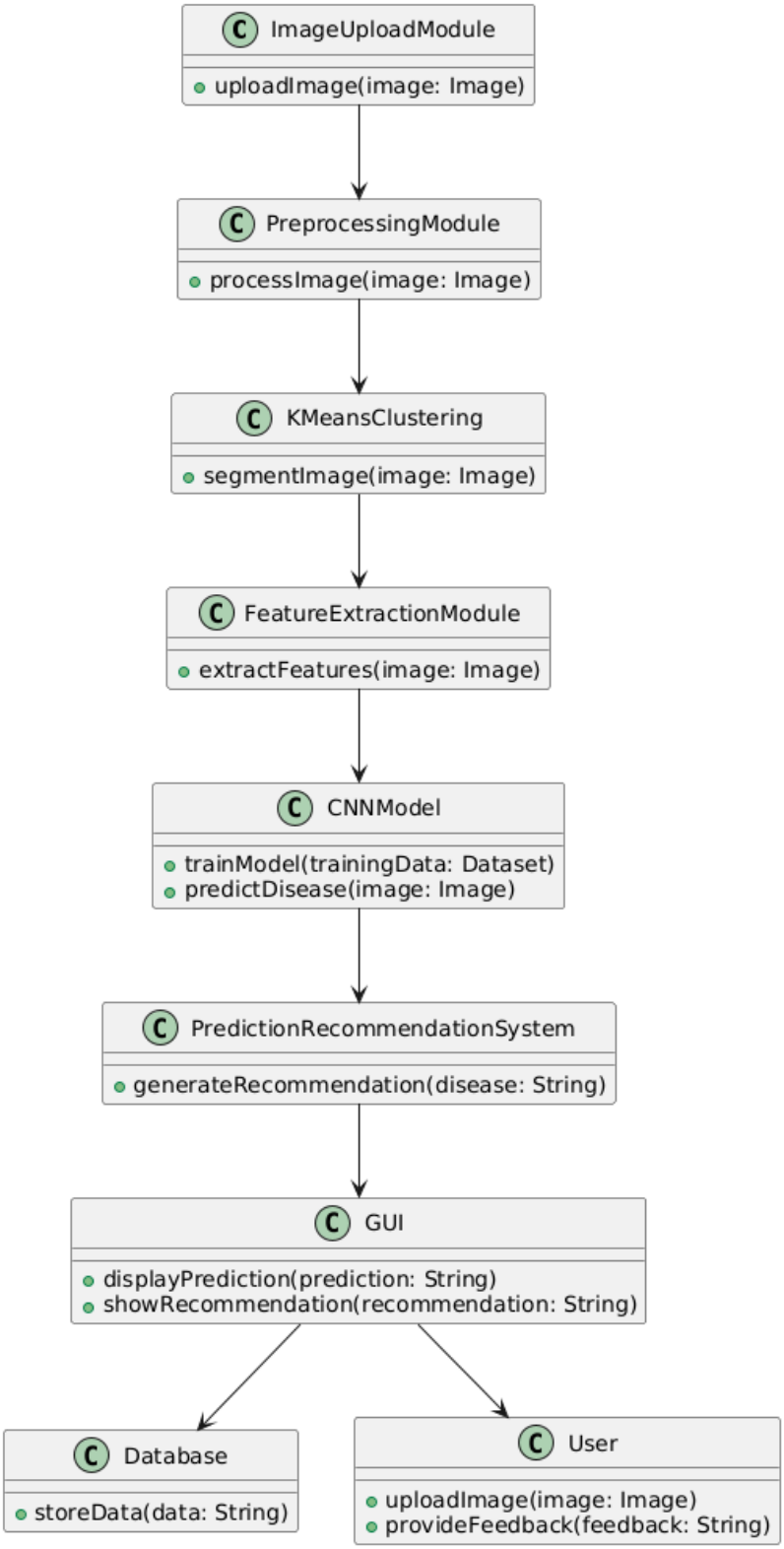

Figure 7 : Class diagram

The class diagram represents the key components of the Rose Plant Disease Detection and Prevention system. It outlines the different classes involved, their responsibilities, and how they interact with each other. Here is a breakdown of each class and its role in the system:

1. **ImageUploadModule:**
    - Responsibilities: Handles the image upload process. The user uploads an image of the rose plant to the system.
    - Methods:
        - uploadImage(image: Image): Receives the uploaded image for processing.
    - Interaction: Sends the uploaded image to the PreprocessingModule.
2. **PreprocessingModule:**
    - Responsibilities: Preprocesses the image to prepare it for analysis. This could include steps like resizing, grayscale conversion, and noise reduction.
    - Methods:
        - processImage(image: Image): Applies various preprocessing techniques on the uploaded image.
    - Interaction: Sends the processed image to KMeansClustering.
3. **KMeansClustering:**
    - Responsibilities: Segments the image using K-means clustering to isolate different areas (diseased and healthy parts).
    - Methods:
        - segmentImage(image: Image): Segments the image into clusters, identifying regions of interest (e.g., diseased areas).
    - Interaction: Sends the segmented image to FeatureExtractionModule.
4. **FeatureExtractionModule:**
    - Responsibilities: Extracts key features from the image, such as texture patterns, using methods like the Gray-Level Co-occurrence Matrix (GLCM).
    - Methods:

        - extractFeatures(image: Image): Extracts relevant features from the processed and segmented image for disease detection.
    - Interaction: Sends the extracted features to CNNModel.

5. **CNNModel:**
    - Responsibilities: A deep learning model used for disease classification based on the extracted features. It uses the features to identify the type of disease affecting the rose plant.
    - Methods:
        - trainModel(trainingData: Dataset): Trains the model using a dataset of labeled images.
        - predictDisease(image: Image): Predicts the disease in a given image using the trained model.
    - Interaction: Sends the disease prediction to PredictionRecommendationSystem.

6. **PredictionRecommendationSystem:**
    - Responsibilities: Based on the disease prediction, it generates treatment recommendations for the user (e.g., farmers or gardeners) to address the detected disease.
    - Methods:
        - generateRecommendation(disease: String): Provides a recommendation for the treatment of the detected disease.
    - Interaction: Sends the treatment recommendation to the GUI.

7. **Database:**
    - Responsibilities: Stores data such as disease predictions, treatment recommendations, and feedback from users.
    - Methods:
        - storeData(data: String): Stores relevant data (e.g., predictions, feedback) in the database.
    - Interaction: Interacts with GUI to store feedback and prediction data.

8. **GUI:**
    - Responsibilities: The graphical user interface (GUI) that allows users to interact with the system. Users can upload images, view disease predictions, and receive treatment recommendations.
    - Methods:
        - displayPrediction(prediction: String): Displays the predicted disease to the user.
        - showRecommendation(recommendation: String): Displays treatment recommendations to the user.
    - Interaction: Receives feedback and interactions from the User and communicates with Database for data storage.
9. **User:**
    - Responsibilities: The user (e.g., farmer, gardener) who interacts with the system to upload images, view disease predictions, and provide feedback.
    - Methods:
        - uploadImage(image: Image): Allows the user to upload a rose plant image for disease detection.
        - provideFeedback(feedback: String): Allows the user to provide feedback on the predictions and recommendations.
    - Interaction: Interacts with GUI to upload images and provide feedback.

Relationships:

- ImageUploadModule sends the uploaded image to PreprocessingModule for further processing.
- PreprocessingModule forwards the processed image to KMeansClustering for segmentation.
- KMeansClustering sends the segmented image to FeatureExtractionModule to extract features for analysis.

- FeatureExtractionModule passes the extracted features to CNNModel, which uses them for disease prediction.
- CNNModel sends the disease prediction result to PredictionRecommendationSystem, which generates appropriate treatment recommendations.
- PredictionRecommendationSystem forwards these recommendations to GUI, which displays them to the user.
- The GUI interacts with the User, allowing them to upload images and provide feedback. It also stores user data (such as predictions and feedback) in the Database.

This diagram and description encapsulate the system's flow, from image upload to disease prediction and recommendation generation, allowing for efficient and automated disease detection and prevention for rose plants.

4.2.4 Sequence Diagram

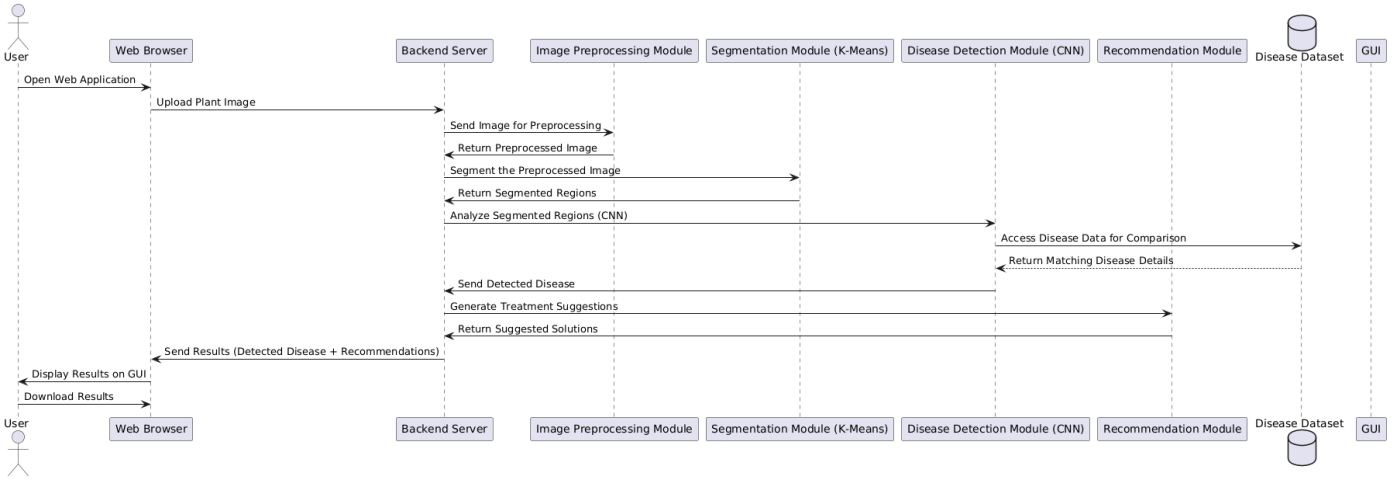

Figure 8 : Sequence Diagram

1. User Interaction:

- User → Image Upload Module: The User uploads an image of the rose plant (or parts of it, like leaves or flowers) to the Image Upload Module via a GUI.

2. Preprocessing:

- Image Upload Module → Preprocessing Module: The image is passed to the Preprocessing Module, where grayscale conversion is performed. This reduces the complexity of the image and helps focus on textural details like spots, powdery patches, or mold patterns.

- Preprocessing Module → K-means Clustering: The processed image (grayscale) is passed to the K-means Clustering algorithm. K-means groups similar pixels into clusters (healthy regions, diseased regions, and background).

3. Feature Extraction:

- K-means Clustering → Feature Extraction Module: The segmented image regions are passed to the Feature Extraction Module, which calculates texture features using the Gray-Level Co-occurrence Matrix (GLCM).

4. Disease Classification:

- Feature Extraction Module → CNN Model: The extracted features are passed to the CNN Model (Separable Neural Network), which classifies the image based on patterns indicative of specific diseases (e.g., Black Spots, Powdery Mildew, Downy Mildew, or Botrytis Blight).
- CNN Model → Prediction & Recommendation System: Once the disease is classified, the CNN Model sends the prediction (disease type) to the Prediction & Recommendation System for treatment suggestions.

5. Storing and Retrieving Data:

- CNN Model → Database: The prediction, along with additional data (e.g., disease type, severity), is stored in the Database for future reference and continuous model improvement.

6**. Treatment Recommendation:**

- Prediction & Recommendation System → GUI: Based on the disease classification, the Prediction & Recommendation System sends recommendations for treatment and prevention to the GUI (e.g., pesticide suggestions, irrigation changes).
- GUI → User: The GUI displays the disease prediction and treatment recommendations to the User.

7. User Feedback (Optional):

- User → GUI: The User may provide feedback on the diagnosis or treatment, or upload additional images for further analysis.
- GUI → Database: The user's feedback is logged in the Database for continuous monitoring and improving the system's accuracy.

Diagram Elements:

- Lifelines for each component (User, Image Upload Module, Preprocessing Module, K-means Clustering, Feature Extraction Module, CNN Model, Prediction & Recommendation System, GUI, Database).
- Activation Boxes to indicate the active state of each component during processing.
- Messages show the flow of data, such as image upload, preprocessing, feature extraction, classification, and prediction.
- Return Messages indicate the output at each stage, such as the prediction results and recommendations.

Key Interactions:

1. User → Image Upload Module: Uploads image for analysis.
2. Image Upload Module → Preprocessing Module: Processes the image (grayscale conversion).
3. Preprocessing Module → K-means Clustering: Segments image for diseased regions.
4. K-means Clustering → Feature Extraction Module: Extracts features using GLCM.
5. Feature Extraction Module → CNN Model: Classifies the disease.
6. CNN Model → Prediction & Recommendation System: Sends disease prediction.
7. Prediction & Recommendation System → GUI: Sends treatment recommendation.
8. GUI → User: Displays prediction and recommendation to the user.
9. User → GUI: Optionally provides feedback.
10. GUI → Database: Stores user feedback and model data.

This sequence diagram represents the workflow from image upload to prediction, classification, and treatment recommendation, providing an efficient system for early rose plant disease detection and prevention.

5. METHODOLOGY AND TESTING

5.1 Module Description:

Rose plant disease prediction project, you've collected a dataset comprising images of four common rose plant diseases: Black Spots, Powdery Mildew, Downy Mildew, and Botrytis Blight. Each of these diseases can manifest in different parts of the rose plant, including the flower, stem, and leaves.

- Black Spots are typically seen as dark, circular spots on the leaves and sometimes on the stems.
- Powdery Mildew is characterized by a white or grayish powdery substance, often appearing on the upper surfaces of leaves, buds, and flowers.
- Downy Mildew manifests as yellowish patches on leaves with a fuzzy appearance on the undersides.
- Botrytis Blight affects blossoms, causing browning and blight in flowers, and sometimes affecting stems and leaves with gray mold.

5.1.1 Data Preprocessing

For accurate disease classification, preprocessing is a critical step. You've opted for grayscale conversion as a preprocessing method. This simplifies the image data by reducing its color information, converting each pixel to a shade of gray, and keeping the necessary details for detecting disease symptoms. Grayscale conversion helps focus on textural details like spots, powdery patches, or mold patterns, which are vital for identifying diseases.

5.1.2 Segmentation with K-Means Clustering

To isolate the diseased areas, you've implemented K-means clustering for segmentation. K-means clustering groups pixels in the image into distinct clusters based on their intensity values. This step enables you to segment different regions of the rose plant, such as healthy parts, diseased parts, or background elements. By applying K-means clustering, you're effectively separating the region of interest (e.g., areas with spots or powdery patches) for further analysis.

Bitwise operations in image processing refer to operations performed on binary (bit-level) representations of pixel values. These operations are particularly useful when you want to isolate specific regions of an image, create masks, or merge images. The bitwise methods provided by OpenCV include AND, OR, XOR, and NOT operations, which operate on the pixel values of two images or an image and a mask.

In the context of disease detection in rose plants, bitwise methods can be used to mask certain parts of an image (like the healthy regions) and focus on the area of interest (e.g., diseased spots). The goal of applying bitwise methods is to extract relevant regions from an image by filtering out unwanted areas.

K-Means Clustering

The algorithm for K–means Clustering:

1. Pick the center of the K cluster, either randomly or based on some heuristic.

2. Assign each pixel in the image to the cluster that minimizes the distance between the pixel and the cluster center.

3. Again compute the cluster centers by averaging all of the pixels in the cluster. Repeat steps 2 and 3 until convergence is attained. Leaf image segmentation using k means clustering algorithm.

$$J(V) = \sum_{i=1}^{c} \sum_{j=1}^{c_i} \left(\left\| x_i - v_j \right\| \right)^2$$

Figure 9 :Equation for K-Means

where,

' |xi- vj |' is the Euclidean distance between xi and vj.

'ci' is the number of data points in ith cluster.

'c' is the number of cluster centers.

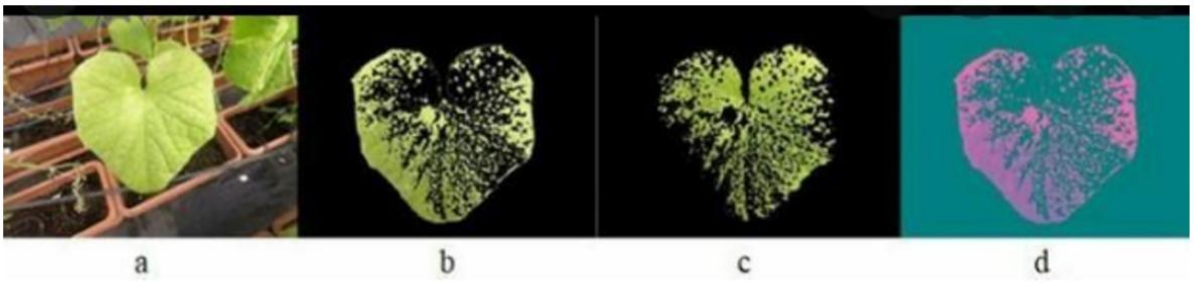

Figure 10 : Image Segmentation using K-means Clustering

a. This is the normal leaf image which is used for classification.

b. In this image we are identifying disease effected part.

c. Increasing the brightness of the effected part.

d. In this image we are highlighting the effected part by the color.

5.1.2 Feature Extraction

The Gray-Level Co-Occurrence Matrix (GLCM) is a powerful tool used in image processing to capture the texture of an image. It calculates how often pairs of pixel values (gray levels) occur in a specific spatial relationship, typically at a certain distance and orientation. The resulting matrix describes the frequency of these co-occurring pixel intensities and provides a statistical representation of texture within the image.

5.1.3 Training with Separable Neural Networks

Your training model is based on a separable neural network, Separable neural networks reduce computational complexity by applying depth wise separable, which split the process of filtering and combining data. This architecture is beneficial for training on image datasets, as it requires fewer parameters and operations, thus speeding up the training process while maintaining high accuracy. The model learns to recognize the distinct visual features of the four rose diseases from the segmented images.

Convolutional Neural Network Design :

The construction of a convolutional neural network is a multi-layered feed-forward neural network, made by assembling many unseen layers on top of each other in a particular order. It is the sequential design that gives permission to CNN to learn hierarchical attributes. In CNN, some of them are followed by grouping layers and hidden layers are typically convolutional layers followed by activation layers. The pre-processing needed in a ConvNet is kindred to that of the related pattern of neurons in the human brain and was motivated by the organization of the Visual Cortex. Aconvolution neural network has multiple hidden layers that help in extracting information from an image.

The four important layers in CNN are:

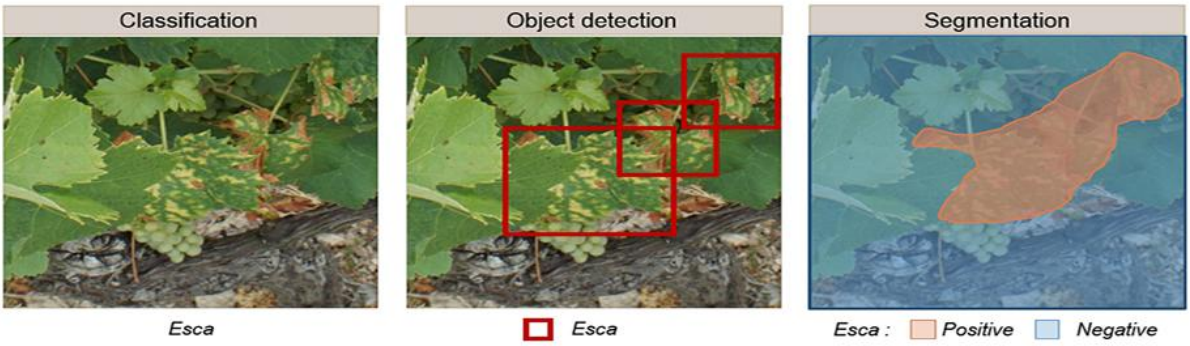

Figure 11 : classification, object detection, segmentation of image

$$x_{i,j}^{l} = \sum_{m}\sum_{n} w_{m,n}^{l} o_{i+m,j+n}^{l-1} + b_{i,j}^{l}$$

$$o_{i,j}^{l} = f(x_{i,j}^{l})$$

$$\delta_{i,j}^{l} = \frac{\partial E}{\partial x_{i,j}^{l}}$$

$$\frac{\partial E}{\partial x_{i',j'}^{l}} = \sum_{m=0}^{k_1-1}\sum_{n=0}^{k_2-1} \delta_{i'-m,j'-n}^{l+1} w_{m,n}^{l+1} f'\left(x_{i',j'}^{l}\right)$$

$$\frac{\partial E}{\partial w_{m',n'}^{l}} = \sum_{i=0}^{H-k_1}\sum_{j=0}^{W-k_2} \delta_{i,j}^{l} o_{i+m',j+n'}^{l-1}$$

Figure 12 : Equation for convolution neural network

Pooling layer

It is possible to reduce the size of an input volume by using both CONV layers with a stride greater than 1 (as we have already seen) and POOL layers. POOL layers are typically added between neighboring CONV layers in CNN architectures:

CONVERSION: RELU: POOL: FCINPUT: CONVERSION: RELU: POOL

Convolution layer

The foundational component of a convolutional neural network is the CONV layer. The parameters of the CONV layer are made up of K learnable filters, or "kernels," each of which has a width and a height and is almost invariably square. Although these filters are modest (in terms of their spatial dimensions), they cover the entire volume's depth.

ReLUlayer

We apply a nonlinear activation function, such as ReLU, ELU, or any of the various Leaky ReLU versions, after each CONV layer in a CNN. Since ReLU activations are most frequently employed, activation layers are typically denoted as RELU in network diagrams. However, we may also just write ACT to indicate that an activation function is being used inside the network design.

Machine Learning Concepts:

Now that we understand some of the perils of polynomial data fitting in the large sample spaces involved in machine learning, we revisit some of the basic concepts we have introduced quite informally so far. These include different types of machine learning problems, the concept of learning a regressor or a classifier from data, and the curse of dimensionality.Machine learning develops algorithms that discover patterns in data. In addition to the problems mentioned in the last set of notes, we consider the following examples of classification, regression, and clustering, with the aim of tightening some of the basic definitions and introducing some notation we will use throughout the course.

Classification:

The US Postal Service (USPS) uses digit recognition, a machine learning technique, to read handwritten ZIP codes on envelopes. Many thousands or even millions of images of the digit ’0’ are provided, together with the label ’0’, and similarly for the other nine digits. These images are provided by the USPS, and someone painstakingly looks at each image and records a label for it in a file. The machine learning algorithm must identify some commonalities among all images of the same digit (these commonalities are the “pattern” in the data) so that if a previously unseen image shows up without a label, it can be automatically labeled without human intervention. The way the machine learning algorithm captures the “pattern” in practice is by computing a function that takes an image as input and produces the appropriate label as output.

Regression:

YouTube may be interested in an automatic method to predict the median age of the viewers of each given video clip, so that it can target ads it shows with each video more specifically and cost-effectively. In this scenario, YouTube may collect data through surveys, perhaps by asking viewers to state their ages voluntarily when they access a new clip. Individual responses may not be very reliable, but if there are enough of them per video clip, the empirical median might be an accurate summary. The machine learning algorithm must identify some pattern that connects properties of

each video to the median age of its viewers. Once that pattern is found, it can be usedto predict median ages of viewers without including the annoying survey.

Clustering:

When a color image is to be sent over an expensive channel or displayed on an inexpensive device, it may not be possible to preserve all the original colors (because doing so would be too expensive, in the first case, or because the device is unable to, in the second). A common solution is to fix the number K of allowable colors to some small integer, and then map each color in the original image to the most similar of the K allowable ones. This requires building a colormap, that is, a table of K allowable colors. Different images have different color distributions, and therefore may require different colormaps for optimal rendering. This color quantization problem can be viewed as one of clustering if the colormap is computed from a subset of the pixels in the original image (as is typically done to save computation effort) and is then applied to all the pixels: The N colors x1, . . . , xN in the sample are viewed as an unlabeled training set X (not T, because there are no labels), and the goal of clustering is to group these colors into K N groups such that colors within a group are similar to each other and colors in different groups are different from each other. Once this task is accomplished, one can select a representative color for each group to form the colormap, and map each of the colors in the original image (typically many more than N) to the nearest of the K representatives (in a metric suitable for colors). Here, the "pattern" in the data is some natural grouping of colors. These examples stem from image or video analysis problems because my main area of research is computer vision. There are many more application domains for machine learning.

5.1.4 Prediction via Graphical User Interface (GUI)

Once the model is trained, the prediction process is facilitated through a Graphical User Interface (GUI). This interface allows users to upload images of rose plants and receive disease predictions based on the model's analysis. The GUI provides a user-friendly platform for interaction, making it accessible for farmers, gardeners, or agricultural experts to quickly diagnose diseases and take preventive measures. Through the GUI, users can visualize the predicted disease, obtain recommendations for treatment, and monitor the health of rose plants in real time.

This workflow, from data collection to prediction, offers an efficient and scalable solution for early disease detection in rose plants, promoting healthier cultivation practices. By combining technology-driven disease prediction with these preventive practices, rose plant growers can effectively manage and mitigate the impact of plant diseases, ensuring long-term sustainability and productivity.

5.1.5 Image Processing Libraries

OpenCV (cv2): Used for basic image processing tasks such as resizing, cropping, and filtering images.

Installation: pip install opencv-python

Usage: Loading and preprocessing images (converting to grayscale, thresholding, etc.).

Pillow (PIL): An image processing library that helps with opening, manipulating, and saving images.

Installation: pip install Pillow

Usage: Image loading, manipulation (resize, rotate), and format conversion.

5.1.6 Machine Learning and Deep Learning Libraries

TensorFlow: A popular deep learning framework for creating, training, and deploying models, especially separable neural network

Installation: pip install TensorFlow

Usage: Building and training deep learning models for disease classification.

Kera's (integrated in TensorFlow): For higher-level model building, training, and evaluation.

5.1.7 Data Handling and Manipulation

NumPy: For handling numerical data, arrays, and matrix operations. Often used to preprocess images and handle model inputs/outputs.

5.1.8 Visualization Libraries

- Matplotlib: For plotting and visualizing the images and data. Helpful for visualizing model performance, such as loss and accuracy plots.
- Installation: pip install matplotlib

Scikit-learn is a powerful and widely-used open-source library in Python for machine learning and data analysis. It provides a variety of efficient tools for classification, regression, clustering, model selection, preprocessing, and dimensionality reduction, among other tasks. It is built on top of NumPy, SciPy, and Matplotlib, making it a great choice for academic and industrial projects alike.

Installation: pip install scikit-learn

5.2 TESTING :

Testing is an essential step in ensuring the robustness, accuracy, and reliability of the **Rose Plant Disease Detection and Prevention** system. Below are the steps for testing the system:

5.2.1 Unit Testing

- **Objective**: To test individual components or functions of the system.
- **Steps**:
    - Test the **image preprocessing** pipeline (e.g., grayscale conversion and resizing) to ensure that the images are processed correctly.
    - Validate the **K-means clustering** algorithm to ensure that the segmentation of diseased and healthy areas is performed accurately.
    - Check the **feature extraction** using GLCM to verify that texture features are being correctly computed.
    - Test the **neural network model**'s forward pass and output to ensure the network is functioning as expected.

5.2.2 Integration Testing

- **Objective**: To test the interaction between different components/modules of the system.
- **Steps**:
    - Validate the end-to-end data flow from **image input** to **disease prediction**. Ensure that the data can be successfully passed through preprocessing, segmentation, feature extraction, and model prediction without any errors.
    - Test the integration of **OpenCV** and **TensorFlow** to ensure smooth communication between image processing and deep learning modules.
    - Ensure that the **GUI** correctly receives and displays results from the model after prediction.

5.2.3 Model Testing

- **Objective**: To test the model's ability to classify and predict plant diseases correctly.
- **Steps**:
    - Split the dataset into **training** and **testing** sets.

- Use the **testing set** to evaluate the trained model.
- Test the model on **new, unseen data** to check if it generalizes well beyond the training set.
- Measure the **performance metrics** (accuracy, precision, recall, F1-score, AUC) on the testing set to ensure that the model is making accurate predictions.
- Test for **overfitting** and **underfitting** by comparing the model's performance on the training and testing datasets.

5.2.4 Segmentation Testing

- **Objective**: To test the segmentation of diseased areas in the images using K-means clustering.
- **Steps**:
  - Test the **K-means clustering** algorithm by applying it to different images and manually inspecting if the segmentation correctly identifies diseased regions like spots, powdery mildew, or mold.
  - Test the **bitwise operations** (AND, OR, NOT) to ensure the accuracy of mask creation and area isolation.

5.2.5 GUI Testing

- **Objective**: To test the functionality and user experience of the graphical user interface.
- **Steps**:
  - Check that users can **upload images** of rose plants via the GUI and that these images are correctly passed to the disease detection system.
  - Test that the **predicted disease results** (e.g., Black Spots, Powdery Mildew) are correctly displayed to the user.
  - Test that the **recommendation system** works (i.e., it provides treatment recommendations for the detected disease).
  - Validate that the GUI is **responsive** and functions across different devices and screen sizes.

5.2.6 End-to-End Testing

- **Objective**: To test the entire system from image upload to disease diagnosis and prevention recommendation.
- **Steps**:
    - Upload a set of **sample images** of rose plants affected by different diseases (Black Spots, Powdery Mildew, etc.) to the GUI.
    - Verify that the system processes these images, classifies the diseases correctly, and displays the appropriate results.
    - Ensure the system provides **disease prevention recommendations** (e.g., pesticide usage or other treatments).
    - Test the overall **user experience**, including navigation, speed, and ease of use.

5.2.7 Stress Testing

- **Objective**: To test how the system performs under heavy load.
- **Steps**:
    - Test the system with **large numbers of concurrent image uploads** to evaluate its performance under load.
    - Check how the system handles a **high volume of image data** (large image sizes, large datasets).
    - Monitor system performance for any slowdowns or crashes, especially during **model prediction** or **segmentation**.

5.2.8 Performance Testing

- **Objective**: To evaluate the speed and efficiency of the system.
- **Steps**:
    - Measure the **time taken** for each image to go through preprocessing, segmentation, and prediction.
    - Test the **inference time** (i.e., the time it takes for the model to classify a new image).

- Evaluate the **memory consumption** during training and inference to ensure the system runs efficiently.

5.2.9 Security Testing

- **Objective**: To ensure the system is secure and prevents vulnerabilities.
- **Steps**:
  - Test the **input validation** on the GUI to ensure that malicious or malformed inputs are not accepted (e.g., checking for image format validation).
  - Ensure the system has proper **access control** if sensitive data is involved.
  - Test for **data privacy** and ensure no personal data or image metadata is leaked.

5.2.10 User Acceptance Testing (UAT)

- **Objective**: To verify that the system meets the needs and expectations of the end users (e.g., farmers, gardeners).
- **Steps**:
  - Have end users test the system with real-world images of rose plants.
  - Gather feedback on the **accuracy** of predictions and the **usefulness** of disease prevention recommendations.
  - Ensure that the system is **easy to use** and that users can understand the results and take action based on them.

5.2.11 Regression Testing

- **Objective**: To ensure that new changes or updates to the system don't break existing functionality.
- **Steps**:
  - After making any modifications or improvements (e.g., model retraining, GUI enhancements), re-test all components to ensure they still function correctly.

5.2.12 Compatibility Testing

- **Objective**: To ensure the system works across various platforms and environments.
- **Steps**:

- Test the system on different **operating systems** (Windows, macOS, Linux).
- Verify that the **GUI** is compatible with multiple browsers (Chrome, Firefox, Safari).
- Check the compatibility of the system with **different devices** (desktop, mobile).

By following these testing steps, you can ensure that your Rose Plant Disease Detection and Prevention system is robust, accurate, efficient, and user-friendly, providing valuable disease detection and prevention recommendations for rose plant cultivation

6.PROJECT DEMONSTRATION

6.1 Dataset:

Figure 13 :Blackspot

Figure 14 : botrytisblight

Figure 15 :Downy Mildew

Figure 16 :Powdery Mildrew

Figure 17 :First Page

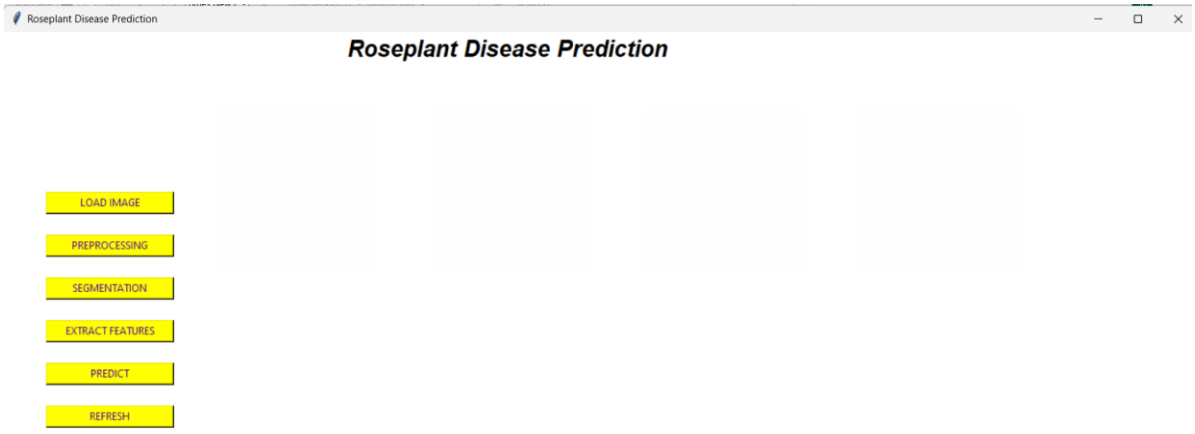

Figure 18 :GUI

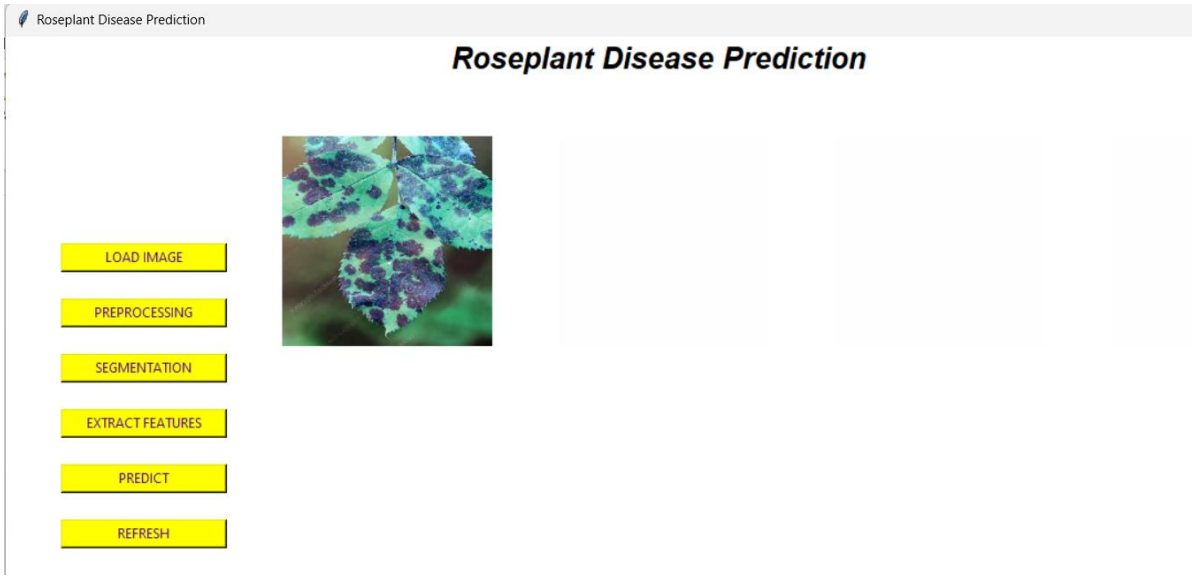

Figure 19 :Loading Image

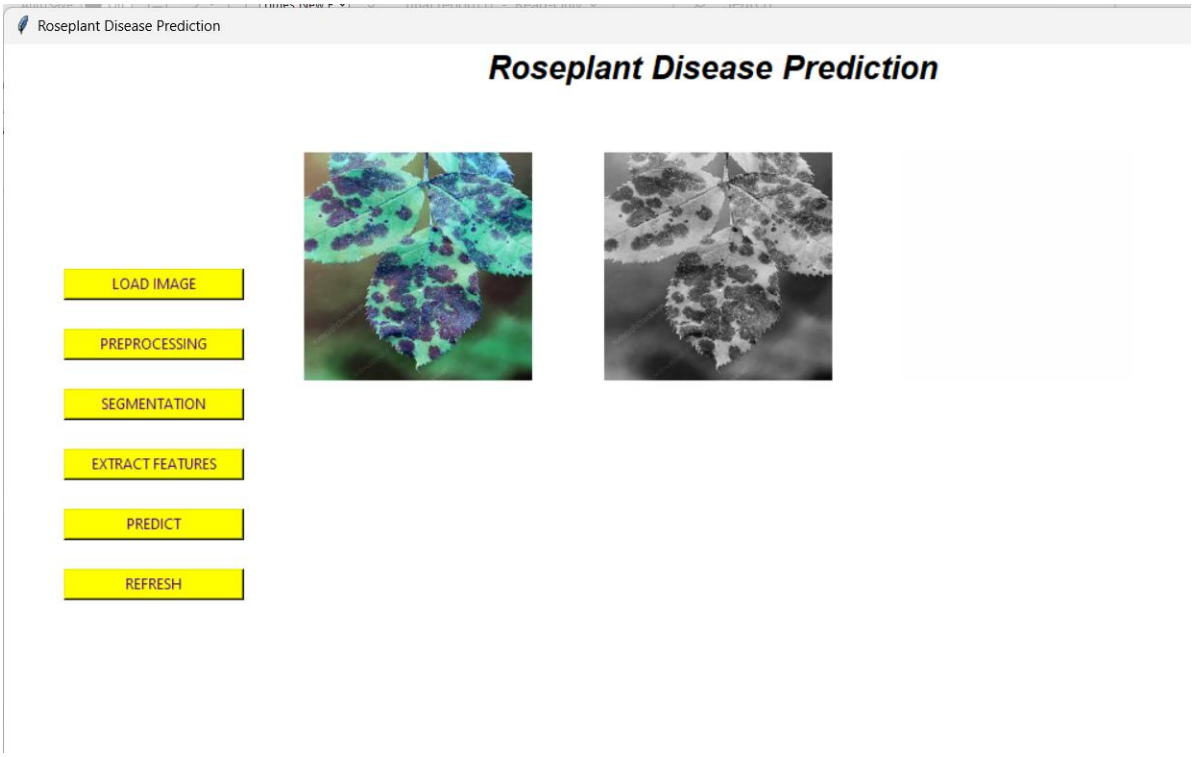

Figure 20 :Preprocessing of Image

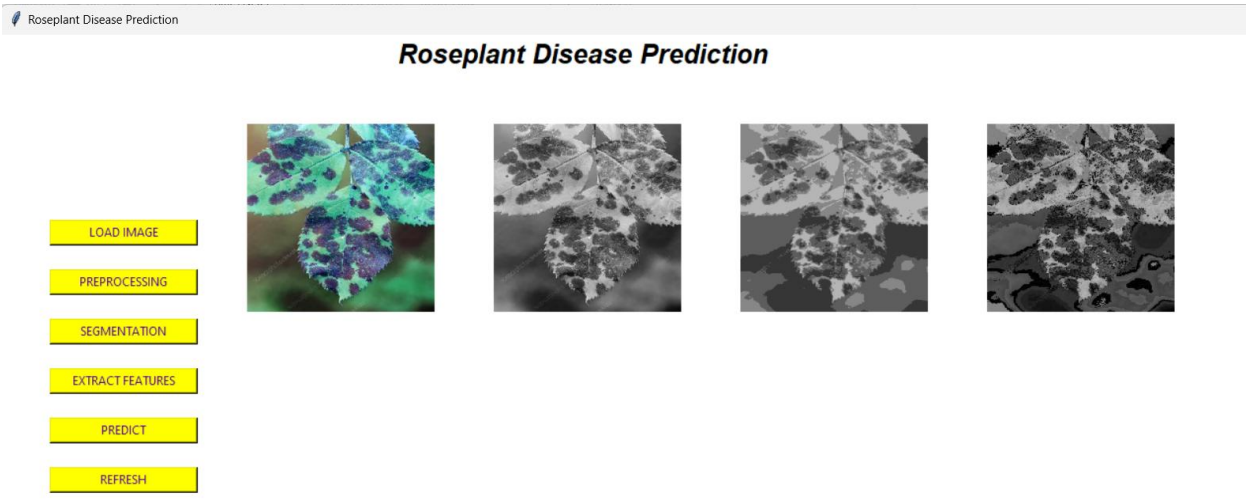

Figure 21 :Segmentation of Image

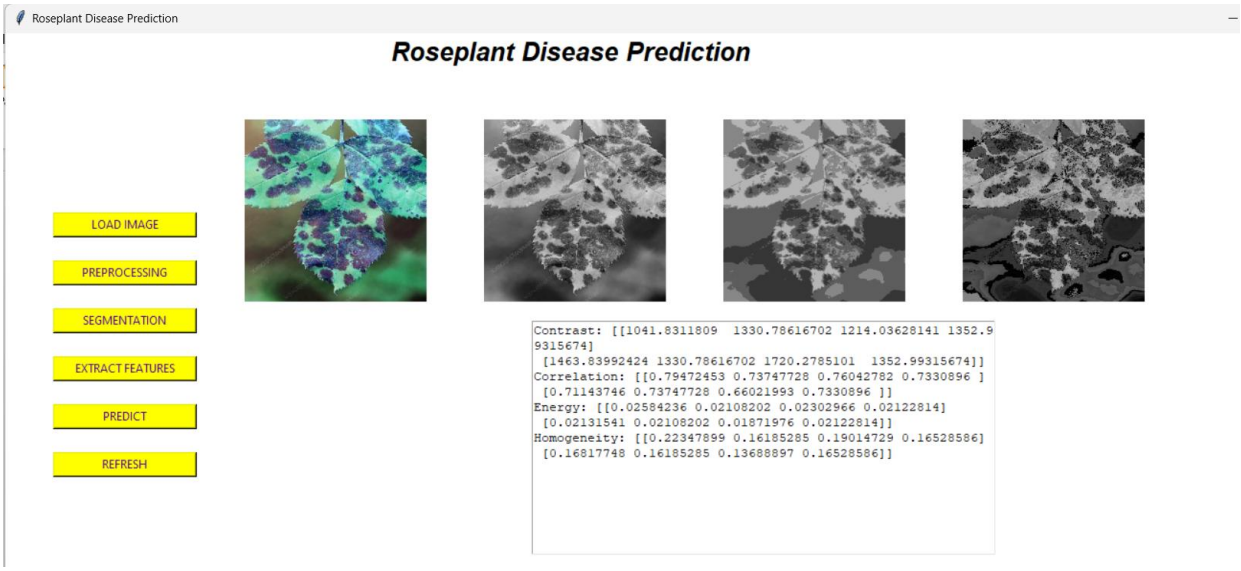

Figure 22 :Extracting Features of Image

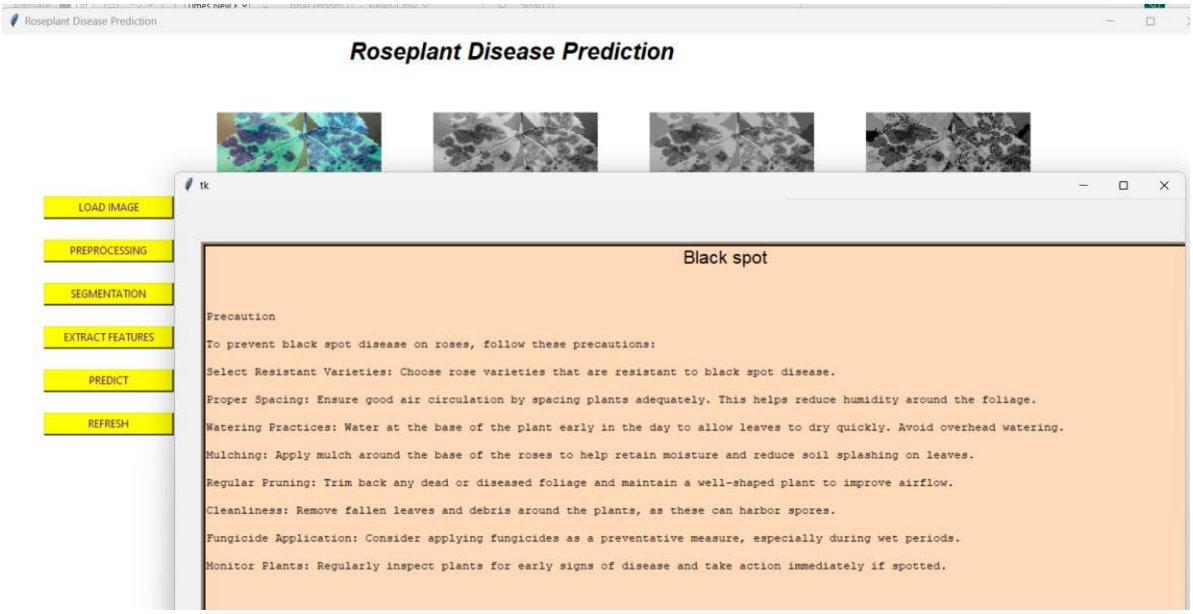

Figure 23 :Predicting Disease

7. RESULT AND DISCUSSION (COST ANALYSIS AS APPLICABLE)

7.1 Results

7.1.1 Output

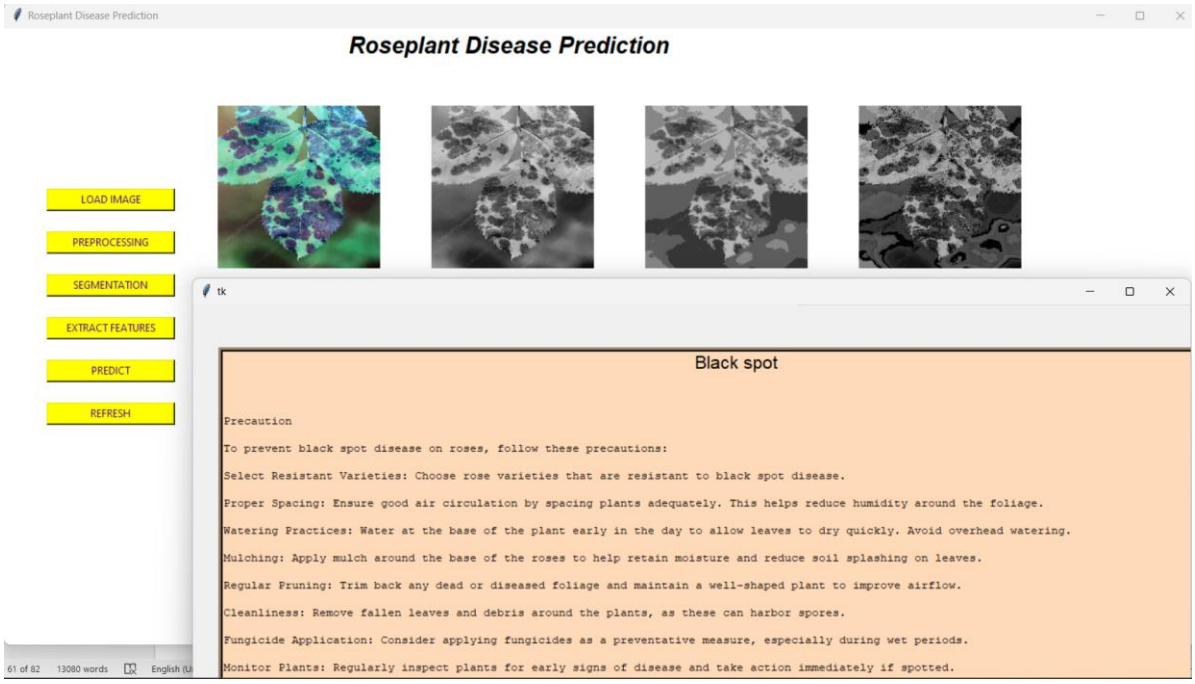

Figure 24 :Final Prediction of the Disease

The image shows a user interface for a rose plant disease prediction system.

Key Components:

- Title Bar: "Rose Plant Disease Detection and Prediction"
- Image Display Area: Displays the uploaded rose plant image in four different views: original, grayscale, segmented, and feature extracted.
- Buttons:
    - LOAD IMAGE: Allows the user to upload a new image.
    - PREPROCESSING: Initiates image preprocessing steps like grayscale conversion and resizing.
    - SEGMENTATION: Segments the image to isolate regions of interest.

- EXTRACT FEATURES: Extracts relevant features from the segmented regions.
- PREDICT: Triggers the disease prediction process using a trained machine learning model.
- REFRESH: Clears the display and resets the system.

- Disease Prediction and Precaution: Displays the predicted disease (in this case, Black Spot) and provides preventive measures to control the disease.

Overall Function:

The system allows users to upload an image of a rose plant, processes the image through various steps, and then predicts the disease affecting the plant. It also provides relevant precautions to prevent the disease.

7.1.2 Accuracy

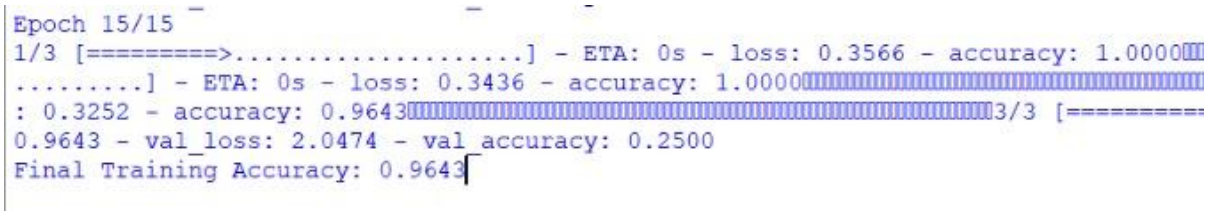

Figure 25 : Accuracy

Accuracy is the most common metric used to evaluate the performance of a classification model. It measures the proportion of correct predictions (both positive and negative) out of all the predictions made. In other words, accuracy answers the question: How many predictions were correct, out of all predictions made?

Accuracy Formula:

(True Positives+ True Negatives) /

(False Negatives False Positives +True Positives + True Negatives)

7.1.3 Accuracy vs epochs

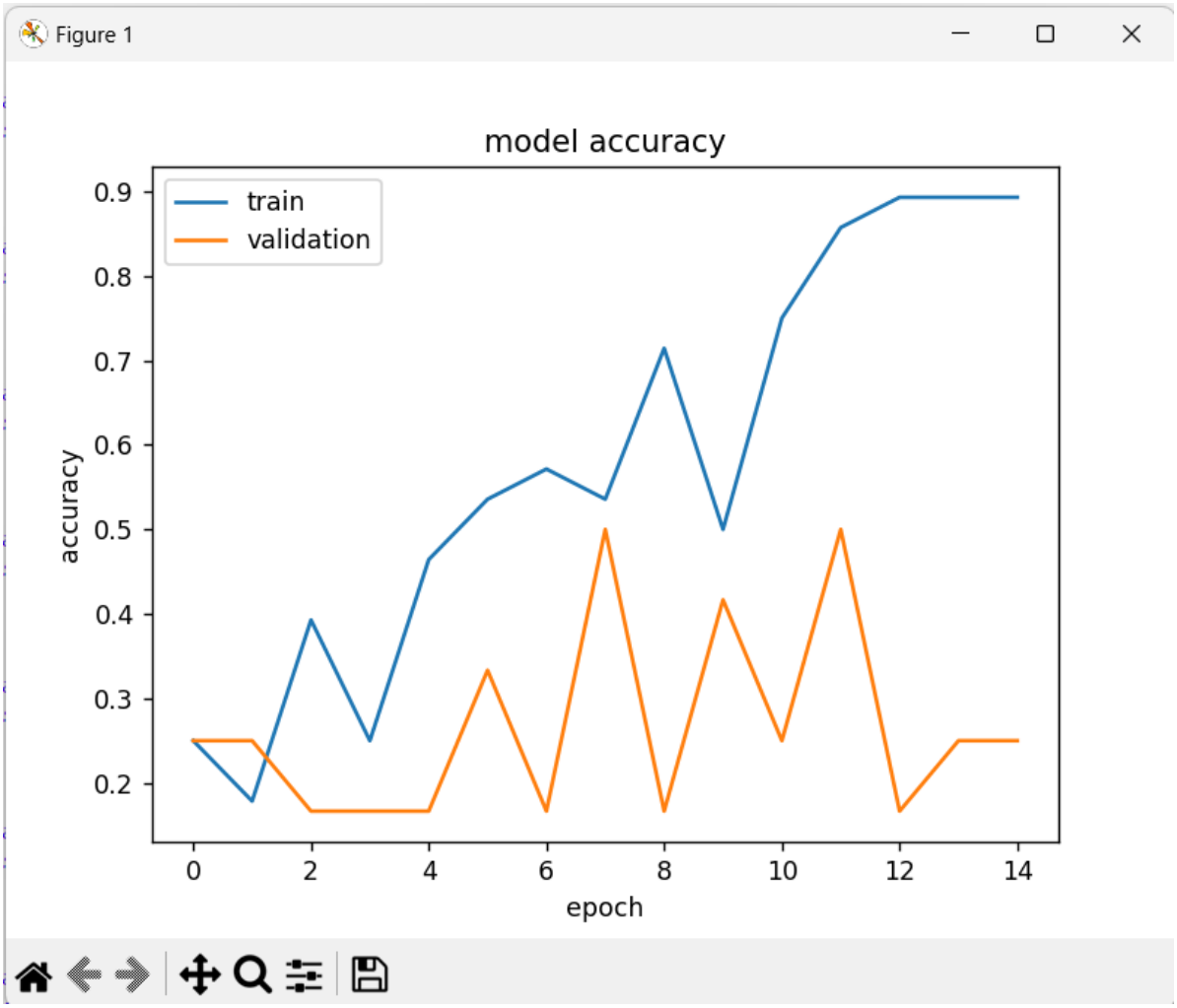

Figure 26 :Accuracy Vs epochs

During the training of a machine learning model, the relationship between accuracy and epochs is crucial for evaluating the model's performance. An epoch refers to one complete pass through the training dataset, during which the model updates its weights. At the beginning of training, accuracy tends to be low because the model starts with random weights and hasn't learned enough about the dataset. As the model progresses through multiple epochs, it adjusts its weights using optimization techniques like gradient descent, leading to an increase in accuracy as it begins to recognize patterns in the data. During the middle epochs, accuracy continues to improve, and the loss (error) typically decreases as the model learns more effectively. However, as training continues into the later epochs, the accuracy curve may plateau, indicating that the model has learned most of the patterns in the data. If training goes on for too many epochs, there is a risk of overfitting, where the model memorizes the training data and fails to generalize to unseen data, resulting in a drop in validation accuracy. Monitoring accuracy vs. epochs helps in identifying the point where the

model has learned enough, and techniques like early stopping can be used to prevent overfitting and ensure that the model generalizes well.

7.1.4 Loss vs epochs

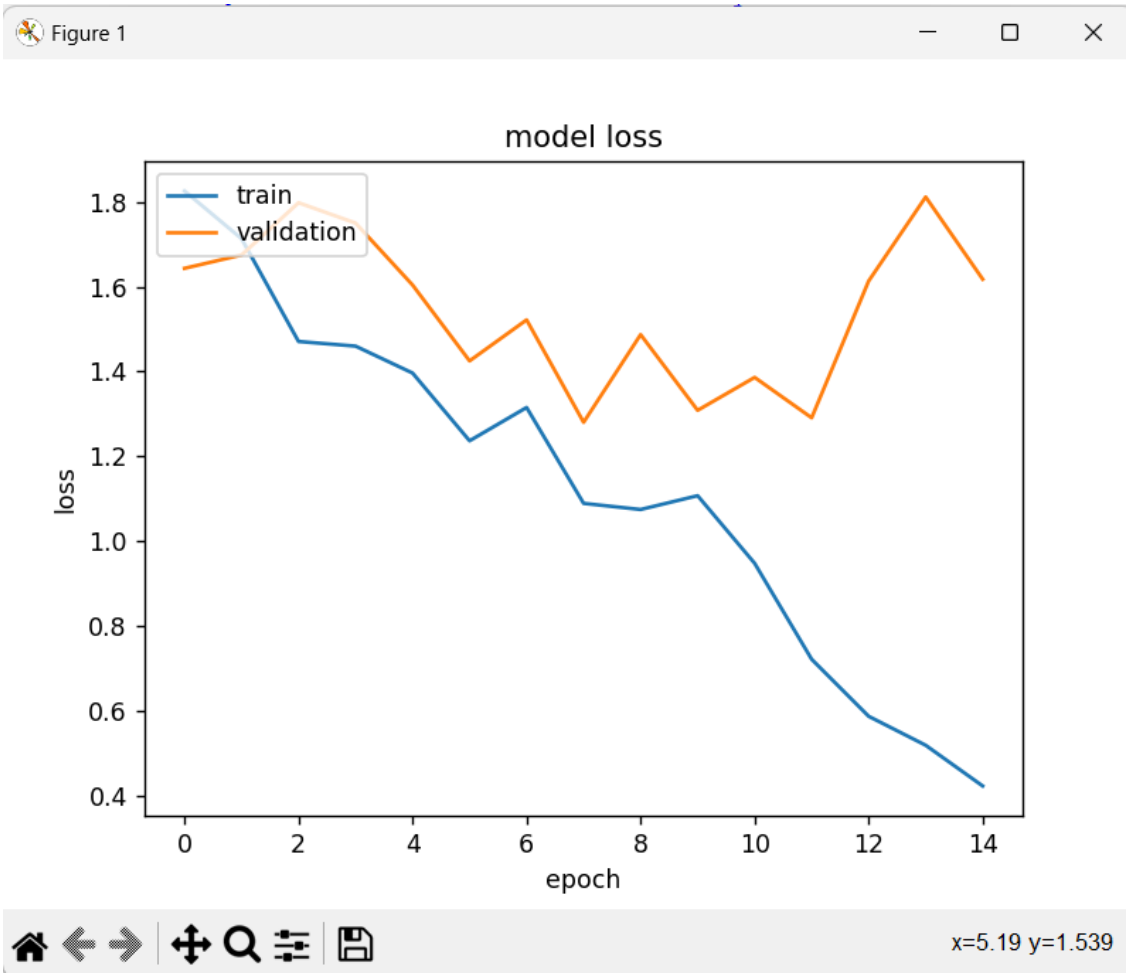

Figure 27 :Loss Vs epochs

The loss vs. epochs curve is a critical tool for evaluating how well a model is learning during training. Loss represents the error or the difference between the model's predicted output and the actual target values. During the early stages of training, the loss is typically high because the model's parameters (weights) are randomly initialized and have not yet been fine-tuned. As training progresses and the model adjusts its parameters through optimization (such as gradient descent), the loss should decrease, indicating that the model is making fewer errors and improving its predictions.

In the middle of the training process, the loss continues to decrease as the model learns from the data. However, after a certain point, the loss curve may start to flatten, signalling that the model has reached a stage where it is not learning much more from the current data, and

further improvements are minimal. This can also indicate that the model is nearing its optimal state for the given dataset. If training continues for too many epochs, there is a risk of overfitting, where the model starts to memorize the training data rather than learning general patterns, which can result in an increase in loss on validation data, even if the training loss continues to decrease.

The loss vs. epochs graph is used to monitor whether the model is converging toward a solution or if it's suffering from issues like overfitting or underfitting. By analysing this curve, one can decide when to stop training or apply techniques like early stopping to prevent unnecessary computation or performance degradation.

7.1.5 Confusion Matrix

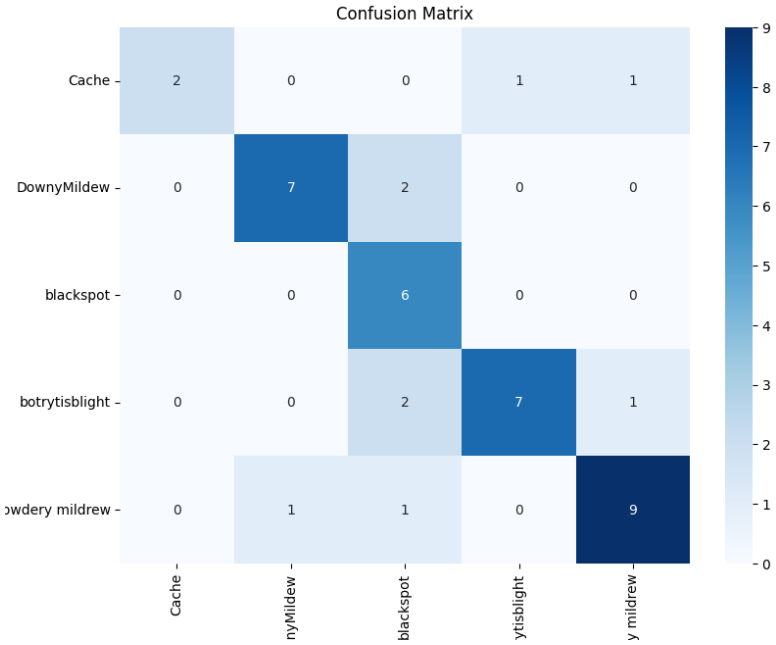

Figure 28 :Confusion Matrix

A confusion matrix is a tool used in machine learning to evaluate the performance of a classification algorithm, particularly in cases where the output is categorical. It is a table that compares the predicted labels with the actual labels in a classification problem, allowing you to visually assess the accuracy and effectiveness of the model.

A confusion matrix typically has the following structure for a binary classification problem (two classes: positive and negative):

	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Table 1: Confusion Matrix Table

Where:

True Positive (TP): The number of instances where the model correctly predicted the positive class.

True Negative (TN): The number of instances where the model correctly predicted the negative class.

False Positive (FP): The number of instances where the model incorrectly predicted the positive class, i.e., the model predicted positive when the actual class was negative.

False Negative (FN): The number of instances where the model incorrectly predicted the negative class, i.e., the model predicted negative when the actual class was positive.

7.1.6 Performance Metrics

Precision:

- Definition: Precision measures the accuracy of the positive predictions made by the model. It answers the question: Of all the instances the model predicted as positive, how many were actually positive?
- Formula: Precision=True Positives / (False Positives + True Positives)
- Interpretation: A higher precision indicates fewer false positives, meaning the model is good at not labelling negative instances as positive.

Recall (Sensitivity or True Positive Rate):

- Definition: Recall measures the ability of the model to identify all relevant positive instances. It answers the question: *Of all the actual positive instances, how many did the model correctly identify?*
- Formula: Recall=True Positives / (True Positives + False Negatives)
- Interpretation: A higher recall means fewer false negatives, and the model is good at identifying all relevant positive cases, but it may also include some false positives.

F1-Score:

- Definition: The F1-Score is the harmonic mean of precision and recall. It is particularly useful when you need to balance both precision and recall. It is ideal when the dataset is imbalanced (i.e., when there are significantly more instances of one class than the other).
- Formula: F1-Score=2×(Precision × Recall / Precision + Recall)
- Interpretation: The F1-Score provides a single metric that balances the trade-off between precision and recall. A high F1-Score means both precision and recall are high.

	precision	recall	f1-score	support
Cache	1.00	0.50	0.67	4
DownyMildew	0.88	0.78	0.82	9
blackspot	0.55	1.00	0.71	6
botrytisblight	0.88	0.70	0.78	10
powdery mildrew	0.82	0.82	0.82	11

Figure 29 :Classification Metrics

7.2 Cost Analysis

Data Collection & Storage Costs

- **Data Collection:**
    - Datasets for disease images are required (Black Spots, Powdery Mildew, Downy Mildew, Botrytis Blight, and healthy plant datasets).
    - Cost can be incurred if datasets need to be purchased or if you have to curate the dataset from various sources.
    - Estimated cost: $0 - $500 (depending on dataset licensing or collection methods).
- **Data Storage:**
    - Cloud storage or local server to store the image datasets, models, and processed data.
    - Cloud providers like AWS S3 or Google Cloud Storage.
    - Estimated cost: $20 - $100 per month (depending on storage size and provider).

Image Processing & Preprocessing Costs

- **Software & Libraries:**
    - OpenCV, TensorFlow, Keras, NumPy, Pillow, Matplotlib, Scikit-learn, and other required Python libraries are mostly open-source and free.
    - Cost: Free, unless using specialized paid tools for enhanced performance.
- **Processing Time:**
    - Image preprocessing (grayscale conversion, K-means clustering, segmentation, bitwise operations, feature extraction) and training the model may require significant computational resources.
    - Using cloud platforms (AWS, Google Cloud) or local GPUs for training and data processing.
    - Estimated cost: $50 - $500 (if using cloud-based computation).

Model Training

- **Hardware:**
    - GPU or cloud-based instances for deep learning model training (e.g., using AWS EC2 instances or Google Colab Pro for cost efficiency).
    - Required for training the Separable CNN model.
    - Estimated cost: $100 - $300 (depending on cloud instance usage or hardware you have).
- **Software**:
    - The training is done using TensorFlow/Keras, which are open-source.
    - Cost: Free for the libraries used.

Development & Implementation Costs

- **Development Team:**
    - Cost of developers, data scientists, or contractors working on the project.
    - If the project is being done individually, there may be no additional cost here.
    - Estimated cost: $0 - $3,000 depending on hours worked and hourly rates.
- **Graphical User Interface (GUI):**
    - Development of GUI for user interaction, allowing farmers or experts to upload images and receive predictions.
    - Cost: $100 - $500 for UI/UX development, if you hire a developer or use paid tools.

Testing and Validation Costs

- **Testing:**
    - Validation of model performance using the test dataset.
    - Costs for obtaining real-world data, if necessary, to test accuracy and reliability.
    - Cost: $0 - $300 (if using external data sources for testing or validating the model's effectiveness).

Maintenance and Updates

- **Model Updates:**
    - Periodic updates to the model, retraining with new data, or adjusting for new diseases or image types.
    - Cost: $50 - $200 per month for cloud resources and maintenance.

Deployment Costs

- **Web Hosting:**
    - Hosting the application on a cloud platform (AWS, Google Cloud, Heroku).
    - Costs may vary based on traffic, number of users, and server load.
    - Cost: $50 - $200 per month depending on usage and server specifications.
- **APIs/Additional Services:**
    - If using any external APIs for image recognition or cloud-based processing, there may be additional costs for API usage.
    - Cost: $10 - $100 per month depending on the API service.

8.CONCLUSION

The Rose Plant Disease Detection and Prevention project exemplifies the convergence of artificial intelligence, image processing, and agricultural practices to address a critical issue in modern farming: plant disease management. By automating the identification of common rose plant diseases, this project enhances the ability of farmers to detect problems early, minimizing the need for harmful pesticides and promoting healthier, more sustainable cultivation practices.

The project employs a series of advanced techniques, including image preprocessing, K-means clustering for segmentation, and deep learning with Separable Convolutional Neural Networks (CNNs), which work in harmony to process and analyse images of rose plants. These technologies enable the system to not only identify visible symptoms of diseases like Black Spots, Powdery Mildew, Downy Mildew, and Botrytis Blight but also to classify and segment the diseased regions from healthy ones. The use of the Gray-Level Co-Occurrence Matrix (GLCM) for feature extraction further refines the model's accuracy, capturing detailed texture patterns that are indicative of disease symptoms.

One of the key aspects of the project is its user-centric approach, with a Graphical User Interface (GUI) that allows farmers and agricultural experts to easily upload images, receive diagnoses, and access treatment recommendations. This level of accessibility ensures that the system can be used by individuals with little technical expertise, empowering them to take quick and informed action.

Beyond the immediate impact on rose plant health, the broader implications of this project extend to the entire field of precision agriculture. As more data is collected and the model is continuously updated with new disease types, the system can be scaled to provide insights for a variety of crops, offering an adaptable and long-term solution to global agricultural challenges. Additionally, the project's focus on early-stage detection helps to reduce the environmental impact of pesticide use, as farmers can target treatments more precisely and use fewer chemicals overall.

The integration of machine learning and computer vision in agriculture paves the way for smarter, more efficient farming practices, which are crucial in a world facing challenges such as climate change, population growth, and resource scarcity. This project serves as a model for how technology can revolutionize traditional industries, improving productivity, sustainability, and profitability.

In conclusion, the Rose Plant Disease Detection and Prevention project not only addresses a critical need within agriculture but also represents the potential for future innovation in plant health management. It combines the power of AI and image processing to offer a tangible, practical solution that benefits both small-scale farmers and large agricultural enterprises, contributing to the broader goals of food security, sustainable farming, and ecological preservation. Through continued research, development, and application, such technologies will undoubtedly play a pivotal role in shaping the future of agriculture.

9. REFERENCES

1.Ali-Al-Alvy, M., Khan, G. K., Alam, M. J., Islam, S., Rahman, M., & Rahman, M. S. (2023, April). Rose plant disease detection using deep learning. In *2023 7th International Conference on Trends in Electronics and Informatics (ICOEI)* (pp. 1244-1249). IEEE.

2)IEEE International Conference on Robotics, Automation, Artificial-Intelligence and Internet-of-Things (RAAICON) the author Sahadat Hossain Khan published a paper on Rose Leaf Disease Detection Using CNN

https://drive.google.com/file/d/1oC345xv2t075ik4niAOCB65ttqoVg2YK/view?usp=sharing

3)International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT) (2020) the author Thoma sarkar mahbubur rehman published a paper on Rose diseases recognition using mobilenet.

https://ieeexplore.ieee.org/abstract/document/9254420

4)Creative commons attribution international (2020) the author Paramashivam, Vijaya lakshmi published a paper on Intelligent plant disease identification using machine learning.

https://www.researchgate.net/publication/347325057_Intelligent_Plant_Disease_Identification_System_Using_Machine_Learning

5) IEEE Eurasia Conference on IOT, Communication and Engineering (ECICE) (2019) the author Sammy Miltane, Bobby dioquino published a paper on Plant Leaf Detection and Disease Recognition using Deep Learning.

https://ieeexplore.ieee.org/document/8942686

APPENDIX A – Sample Code

Training the Model:

```
from sklearn.datasets import load_files
from keras.utils import to_categorical
import numpy as np
from glob import glob

tar=5
path='./dataset/'
# define function to load train, test, and validation datasets
def load_dataset(path):
    data = load_files(path)
    files = np.array(data['filenames'])
    targets =to_categorical(np.array(data['target']), tar)
    return files, targets

# load train, test, and validation datasets
train_files, train_targets = load_dataset(path)

test_files=train_files
test_targets = train_targets


burn_classes = [item[10:-1] for item in sorted(glob("./dataset/*/"))]
# print statistics about the dataset
print('There are %d total categories.' % len(burn_classes))
print(burn_classes)
print('There are %s total  images.\n' % len(np.hstack([train_files,
test_files])))
print('There are %d training images.' % len(train_files))
print('There are %d test images.'% len(test_files))



for file in train_files: assert('.DS_Store' not in file)



from tensorflow.keras.preprocessing import image
from tqdm import tqdm

# Note: modified these two functions, so that we can later also read the
inception tensors which
# have a different format
def path_to_tensor(img_path, width=224, height=224):
```

```
    # loads RGB image as PIL.Image.Image type
    img = image.load_img(img_path, target_size=(width, height))
    # convert PIL.Image.Image type to 3D tensor with shape (width, heigth, 3)
    x = image.img_to_array(img)
    # convert 3D tensor to \4D tensor with shape (1, width, height, 3) and
return 4D tensor
    return np.expand_dims(x, axis=0)

def paths_to_tensor(img_paths, width=224, height=224):
    list_of_tensors = [path_to_tensor(img_path, width, height) for img_path
in tqdm(img_paths)]
    return np.vstack(list_of_tensors)

import keras
import timeit

# graph the history of model.fit
def show_history_graph(history):
    # summarize history for accuracy
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'validation'], loc='upper left')
    plt.show()
    # summarize history for loss
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'validation'], loc='upper left')
    plt.show()


# pre-process the data for Keras
train_tensors = paths_to_tensor(train_files).astype('float32')/255
test_tensors = paths_to_tensor(test_files).astype('float32')/255


from tensorflow.keras.layers import Conv2D, MaxPooling2D,
GlobalAveragePooling2D
from tensorflow.keras.layers import Dropout, Flatten, Dense
from tensorflow.keras.models import Sequential
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import ModelCheckpoint

import matplotlib.pyplot as plt
```

```
img_width, img_height = 224, 224
batch_size = 4
epoch=10


img_width, img_height = img_width, img_height
batch_size = 32
samples_per_epoch = 40
validation_steps = 40
nb_filters1 = 32
nb_filters2 = 64
conv1_size = 3
conv2_size = 3
pool_size = 3
lr = 0.0005
from tensorflow.keras import optimizers
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dropout, Flatten, Dense, Activation
from tensorflow.keras.layers import Convolution2D, MaxPooling2D
from tensorflow.keras import callbacks
import time
from keras.layers import SeparableConv2D, Conv2D, Activation, MaxPooling2D,
Flatten, Dense, Dropout

model = Sequential()

# First Convolution Block
model.add(SeparableConv2D(nb_filters1, (conv1_size, conv1_size),
padding='same', input_shape=(img_width, img_height, 3)))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(pool_size, pool_size)))

# First Separable Convolution Layer
model.add(SeparableConv2D(nb_filters2, (conv2_size, conv2_size),
padding='same'))
model.add(Activation("relu"))
'''

# Second Separable Convolution Layer
model.add(SeparableConv2D(nb_filters2, (conv2_size, conv2_size),
padding='same'))
model.add(Activation("relu"))

'''

# Pooling after Separable Convolution layers
model.add(MaxPooling2D(pool_size=(pool_size, pool_size)))

# Fully connected layers
model.add(Flatten())
```

```
model.add(Dense(256))
model.add(Activation("relu"))
model.add(Dropout(0.5))
model.add(Dense(tar, activation='softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer=optimizers.RMSprop(lr=lr),
              metrics=['accuracy'])

hist=model.fit(train_tensors, train_targets ,validation_split=0.3, epochs=15,
batch_size=10)

# Print the accuracy after training
train_acc = hist.history['accuracy'][-1]
#val_acc = hist.history['val_accuracy'][-1]

print(f"Final Training Accuracy: {train_acc:.4f}")
#print(f"Final Validation Accuracy: {val_acc:.4f}")


show_history_graph(hist)


#model.save('color_trained_modelDNN.h5')
model.save('trained_model_DNN1.h5')
```

GUI Implementation:

```
from tkinter import *
import tkinter as tk
import cv2
from tkinter import filedialog, messagebox
import os
import numpy as np
from PIL import ImageTk, Image
from skimage.color import rgb2gray
from skimage.feature import graycomatrix, graycoprops
from tensorflow.keras.models import load_model
from glob import glob
global T, rep


class Window(Frame):
    def __init__(self, master=None):
        Frame.__init__(self, master)
        self.master = master
        self.config(bg="white")
        self.master.title("Roseplant Disease Prediction")
        self.pack(fill=BOTH, expand=1)
```

```
        # Title label
        w = Label(self, text="Roseplant Disease Prediction", fg="black",
bg="#FFFFFF", font="Helvetica 20 bold italic")
        w.pack()
        w.place(x=400, y=0)

        self.image_labels = []  # Initialize the list to hold image labels
        self.create_labels()

        # Button instances
        Button(self, command=self.load, text="LOAD IMAGE", bg="#FFFF00",
fg="#4C0099", activebackground="dark red", width=20).place(x=50, y=200,
anchor="w")
        Button(self, command=self.classification, text="PREDICT",
bg="#FFFF00", fg="#4C0099", activebackground="dark red",
width=20).place(x=50, y=400, anchor="w")
        Button(self, command=self.preprocessing, text="PREPROCESSING",
bg="#FFFF00", fg="#4C0099", activebackground="dark red",
width=20).place(x=50, y=250, anchor="w")
        Button(self, command=self.segmentation, text="SEGMENTATION",
bg="#FFFF00", fg="#4C0099", activebackground="dark red",
width=20).place(x=50, y=300, anchor="w")
        Button(self, command=self.extract_features, text="EXTRACT FEATURES",
bg="#FFFF00", fg="#4C0099", activebackground="dark red",
width=20).place(x=50, y=350, anchor="w")
        Button(self, command=self.refresh, text="REFRESH", bg="#FFFF00",
fg="#4C0099", activebackground="dark red", width=20).place(x=50, y=450,
anchor="w")
##
    def create_labels(self):
        # Create and store labels in the image_labels list
        for i in range(4):  # Assuming you have 4 labels for images
            label = Label(self, borderwidth=15, highlightthickness=5,
height=150, width=150)
            label.place(x=(250 + i * 250), y=90)  # Adjust positions
            self.image_labels.append(label)

        # Display a placeholder logo in the labels
        load = Image.open(r"logo.jfif")
        render = ImageTk.PhotoImage(load.resize((200, 200)))
        for label in self.image_labels:
            label.config(image=render)
            label.image = render

    def load(self, event=None):
        global rep
        rep = filedialog.askopenfilenames()
```

```
        img = cv2.imread(rep[0])
        self.from_array = Image.fromarray(cv2.resize(img, (200, 200)))
        render = ImageTk.PhotoImage(self.from_array)
        self.image_labels[0].config(image=render)
        self.image_labels[0].image = render

    def preprocessing(self, event=None):
        img = cv2.imread(rep[0])
        hsv_img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
        self.from_array = Image.fromarray(cv2.resize(hsv_img, (200, 200)))
        render = ImageTk.PhotoImage(self.from_array)
        self.image_labels[1].config(image=render)
        self.image_labels[1].image = render

    def segmentation(self, event=None):
        img = cv2.imread(rep[0])
        hsv_img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
        gauss = cv2.GaussianBlur(hsv_img, (5, 5), 0)

        # KMeans segmentation
        pixel_values = gauss.reshape((-1, 1))
        pixel_values = np.float32(pixel_values)
        criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 100,
0.2)
        k = 4
        _, labels, centers = cv2.kmeans(pixel_values, k, None, criteria, 10,
cv2.KMEANS_RANDOM_CENTERS)
        centers = np.uint8(centers)
        segmented_image = centers[labels.flatten()].reshape(gauss.shape)

        self.from_array = Image.fromarray(cv2.resize(segmented_image, (200,
200)))
        render = ImageTk.PhotoImage(self.from_array)
        self.image_labels[2].config(image=render)
        self.image_labels[2].image = render

##        # Dilation (optional, can be shown as a fourth image)
##        kernel = np.ones((5, 5), np.uint8)
##        dilated_image = cv2.dilate(segmented_image, kernel, iterations=1)
##        self.from_array = Image.fromarray(cv2.resize(dilated_image, (200,
200)))
##        render = ImageTk.PhotoImage(self.from_array)


        kernel = np.ones((5, 5), np.uint8)

        # Example bitwise operation: AND operation with a kernel
        bitwise_image = cv2.bitwise_and(hsv_img, segmented_image, mask=None)

        # Resize and convert to PIL image
```

```
        self.from_array = Image.fromarray(cv2.resize(bitwise_image, (200,
200)))

        # Update the label with the new image
        render = ImageTk.PhotoImage(self.from_array)
        self.image_labels[3].config(image=render)
        self.image_labels[3].image = render

    def extract_features(self):
        if len(self.from_array.mode) != 1:
            gray_image = self.from_array.convert('L')
        else:
            gray_image = self.from_array

        gray_image = np.array(gray_image)
        distances = [1, 2]
        angles = [0, np.pi / 4, np.pi / 2, 3 * np.pi / 4]
        glcm = graycomatrix(gray_image, distances=distances, angles=angles,
levels=256, symmetric=True, normed=True)

        contrast = graycoprops(glcm, 'contrast')
        correlation = graycoprops(glcm, 'correlation')
        energy = graycoprops(glcm, 'energy')
        homogeneity = graycoprops(glcm, 'homogeneity')

        # Display features
        features = {
            'Contrast': contrast,
            'Correlation': correlation,
            'Energy': energy,
            'Homogeneity': homogeneity
        }

        m = "\n".join([f"{name}: {value}" for name, value in
features.items()])

        self.result_text1 = Text(self, height=15, width=60,background=
"white")
        self.result_text1.place(x=550, y=300)

        self.result_text1.delete(1.0, END)
        self.result_text1.insert(END, m)

    def classification(self, event=None):

        global T, rep
```

```
        from glob import glob
        #from keras.preprocessing import image
        from tensorflow.keras.models import load_model

        # Get the list of classes from the directory structure
        clas1 = [item[10:-1] for item in sorted(glob("./dataset/*/"))]
        print(clas1)
        from tensorflow.keras.preprocessing import image
        #from tqdm import tqdm
        def path_to_tensor(img_path, width=224, height=224):
            print(img_path)
            img = image.load_img(img_path, target_size=(width, height))
            x = image.img_to_array(img)
            return np.expand_dims(x, axis=0)
        def paths_to_tensor(img_paths, width=224, height=224):
            list_of_tensors = [path_to_tensor(img_paths, width, height)]
            return np.vstack(list_of_tensors)
        from tensorflow.keras.models import load_model

        # Load the pre-trained model
        model1 = load_model('trained_model_DNN1.h5')

        # Load and preprocess the test image
        test_tensors1 = paths_to_tensor(rep[0]) / 255

        # Make predictions using the model
        pred1 = model1.predict(test_tensors1)
        print(pred1)
        # Get the index of the predicted class
        predicted_class_index1 = np.argmax(pred1)
        print(predicted_class_index1)

        # Display the predicted class
        res3 = 'Predicted disease is ' + clas1[predicted_class_index1]


        # Display the predicted class in the GUI
##        T = Text(self, height=5, width=40)
##        T.place(x=550, y=600)
##        T.insert(END, res3)
##        self.result_text2 = Text(self, height=15, width=40,background=
"white")
##        self.result_text2.place(x=550, y=300)
##        self.result_text2.delete(1.0, END)
##        self.result_text2.insert(END, res3)



        if predicted_class_index1 == 0:
```

```
            # Path to your file
            yt_file_path = './files/Cache.txt'

            # Read the content of the file
            with open(yt_file_path, 'r', encoding='utf-8') as file:
                file_content = file.read()

            # Create Tkinter window
            root = tk.Tk()
            root.geometry("700x700")  # Set window size

            # Create Text widget with specified height and width
            T = tk.Text(root, bg="#ffdab9", height=35, width=150, bd=5,
relief='sunken')  # Adjust height and width as needed
            T.place(x=30, y=50)  # Set position

            # Create a tag with specified font size
            T.tag_configure("font_size", font=("Helvetica",
16),justify='center')  # Change "Helvetica" and 12 to desired font and size

            # Insert the content of the file into the Text widget
            T.insert(tk.END, file_content.strip())

            # Specify the line number you want to change (0-based index)
            line_number = 0  # Change to your desired line number (e.g., 2
for the third line)

            # Get the start and end indices for the specified line
            start_index = f"{line_number + 1}.0"  # line_number + 1 because
of 0-based index
            end_index = f"{line_number + 1}.end"

            # Apply the tag to that specific line
            T.tag_add("font_size", start_index, end_index)



        if predicted_class_index1 == 1:
            # Path to your file
            yt_file_path = './files/DownyMildew.txt'

            # Read the content of the file
            with open(yt_file_path, 'r', encoding='utf-8') as file:
                file_content = file.read()

            # Create Tkinter window
            root = tk.Tk()
            root.geometry("700x700")  # Set window size
```

```
            # Create Text widget with specified height and width
            T = tk.Text(root, bg="#ffdab9", height=35, width=150, bd=5,
relief='sunken')  # Adjust height and width as needed
            T.place(x=30, y=50)  # Set position

            # Create a tag with specified font size
            T.tag_configure("font_size", font=("Helvetica", 16),
justify='center')  # Change "Helvetica" and 12 to desired font and size

            # Insert the content of the file into the Text widget
            T.insert(tk.END, file_content.strip())

            # Specify the line number you want to change (0-based index)
            line_number = 0  # Change to your desired line number (e.g., 2
for the third line)

            # Get the start and end indices for the specified line
            start_index = f"{line_number + 1}.0"  # line_number + 1 because
of 0-based index
            end_index = f"{line_number + 1}.end"

            # Apply the tag to that specific line
            T.tag_add("font_size", start_index, end_index)


        if predicted_class_index1 == 2:
            # Path to your file
            yt_file_path = './files/blackspot.txt'

            # Read the content of the file
            with open(yt_file_path, 'r', encoding='utf-8') as file:
                file_content = file.read()

            # Create Tkinter window
            root = tk.Tk()
            root.geometry("700x700")  # Set window size

            # Create Text widget with specified height and width
            T = tk.Text(root, bg="#ffdab9", height=35, width=150, bd=5,
relief='sunken')  # Adjust height and width as needed
            T.place(x=30, y=50)  # Set position

            # Create a tag with specified font size
            T.tag_configure("font_size", font=("Helvetica", 16),
justify='center')  # Change "Helvetica" and 12 to desired font and size

            # Insert the content of the file into the Text widget
            T.insert(tk.END, file_content.strip())

            # Specify the line number you want to change (0-based index)
```

```
            line_number = 0  # Change to your desired line number (e.g., 2
for the third line)

            # Get the start and end indices for the specified line
            start_index = f"{line_number + 1}.0"  # line_number + 1 because
of 0-based index
            end_index = f"{line_number + 1}.end"

            # Apply the tag to that specific line
            T.tag_add("font_size", start_index, end_index)



        if predicted_class_index1 == 3:
            # Path to your file
            yt_file_path = './files/botrytisblight.txt'

            # Read the content of the file
            with open(yt_file_path, 'r', encoding='utf-8') as file:
                file_content = file.read()

            # Create Tkinter window
            root = tk.Tk()
            root.geometry("700x700")  # Set window size

            # Create Text widget with specified height and width
            T = tk.Text(root, bg="#ffdab9", height=35, width=150, bd=5,
relief='sunken')  # Adjust height and width as needed
            T.place(x=30, y=50)  # Set position

            # Create a tag with specified font size
            T.tag_configure("font_size", font=("Helvetica", 16),
justify='center')  # Change "Helvetica" and 12 to desired font and size

            # Insert the content of the file into the Text widget
            T.insert(tk.END, file_content.strip())

            # Specify the line number you want to change (0-based index)
            line_number = 0  # Change to your desired line number (e.g., 2
for the third line)

            # Get the start and end indices for the specified line
            start_index = f"{line_number + 1}.0"  # line_number + 1 because
of 0-based index
            end_index = f"{line_number + 1}.end"

            # Apply the tag to that specific line
            T.tag_add("font_size", start_index, end_index)
```

```
        if predicted_class_index1 == 4:
            # Path to your file
            yt_file_path = './files/powdery mildrew.txt'

            # Read the content of the file
            with open(yt_file_path, 'r', encoding='utf-8') as file:
                file_content = file.read()

            # Create Tkinter window
            root = tk.Tk()
            root.geometry("700x700")  # Set window size

            # Create Text widget with specified height and width
            T = tk.Text(root, bg="#ffdab9", height=35, width=150, bd=5,
relief='sunken')  # Adjust height and width as needed
            T.place(x=30, y=50)  # Set position

            # Create a tag with specified font size
            T.tag_configure("font_size", font=("Helvetica", 16),
justify='center')  # Change "Helvetica" and 12 to desired font and size

            # Insert the content of the file into the Text widget
            T.insert(tk.END, file_content.strip())

            # Specify the line number you want to change (0-based index)
            line_number = 0  # Change to your desired line number (e.g., 2
for the third line)

            # Get the start and end indices for the specified line
            start_index = f"{line_number + 1}.0"  # line_number + 1 because
of 0-based index
            end_index = f"{line_number + 1}.end"

            # Apply the tag to that specific line
            T.tag_add("font_size", start_index, end_index)



    def refresh(self):
        for label in self.image_labels:
            label.config(image='')
            label.image = None

        self.result_text1.delete(1.0, END)
        #self.after(2,self.clear_text)

        self.result_text2.delete(1.0, END)
        #self.result_text.delete(1.0, END)
        global rep
        rep = None
```

```
class LoginWindow(Frame):
    def __init__(self, master=None):
        Frame.__init__(self, master)
        self.master = master
        self.config(bg='white')
        self.master.title("Login")

        bg_image = Image.open("rose.jpg")
        bg_render = ImageTk.PhotoImage(bg_image)
        self.background_label = Label(self, image=bg_render)
        self.background_label.image = bg_render
        self.background_label.place(x=0, y=0, relwidth=1, relheight=1)
        self.pack(fill=BOTH, expand=1)
        text_label = Label(self, text="Welcome to the Roseplant Disease
Prediction", bg='white', fg='black', font=("Helvetica", 16))
        text_label.place(x=450, y=200)  # Adjust the position as needed




##        self.login_button = Button(self, text="chatbot",
command=self.login)
##        self.login_button.place(x=650, y=200)

        self.login_button = Button(self, text="upload", command=self.login)
        self.login_button.place(x=650, y=300)

    def login(self):
        self.master.switch_frame(Window)



class MainApplication(Tk):
    def __init__(self, *args, **kwargs):
        Tk.__init__(self, *args, **kwargs)
        self.title("Roseplane disease Prediction")
        self.geometry("1400x720")
        self.current_frame = None
        self.switch_frame(LoginWindow)

    def switch_frame(self, frame_class):
        new_frame = frame_class(self)
        if self.current_frame:
            self.current_frame.destroy()
        self.current_frame = new_frame
        self.current_frame.pack()

if __name__ == "__main__":
```

```
    app = MainApplication()
    app.mainloop()

##root = Tk()
##root.geometry("1400x720")
##app = Window(root)
##root.mainloop()
```

```
from tkinter import *
import tkinter as tk
import cv2
from tkinter import filedialog, messagebox
import os
import numpy as np
from PIL import ImageTk, Image
from skimage.feature import graycomatrix, graycoprops
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.image import load_img, img_to_array

from glob import glob
from keras.preprocessing import image

class Window(Frame):
    def __init__(self, master=None):
        super().__init__(master)
        self.master = master
        self.config(bg="white")
        self.master.title("Roseplant Disease Prediction")
        self.pack(fill=BOTH, expand=1)

        # Initialize instance variables
        self.rep = None
        self.model = load_model('trained_model_DNN.h5')
        self.image_labels = []

        # Title label
        Label(self, text="Roseplant Disease Prediction", fg="black",
bg="#FFFFFF", font="Helvetica 20 bold italic").pack(pady=(10, 0))

        self.create_labels()
        self.create_buttons()

    def create_labels(self):
        for i in range(4):
```

```
            label = Label(self, borderwidth=15, highlightthickness=5,
height=150, width=150)
            label.place(x=(250 + i * 250), y=90)
            self.image_labels.append(label)

        load = Image.open("logo.jfif")
        render = ImageTk.PhotoImage(load.resize((200, 200)))
        for label in self.image_labels:
            label.config(image=render)
            label.image = render

    def create_buttons(self):
        buttons = [
            ("LOAD IMAGE", self.load),
            ("PREPROCESSING", self.preprocessing),
            ("SEGMENTATION", self.segmentation),
            ("EXTRACT FEATURES", self.extract_features),
            ("PREDICT", self.classification),
            ("REFRESH", self.refresh)
        ]
        for i, (text, command) in enumerate(buttons):
            Button(self, command=command, text=text, bg="#FFFF00",
fg="#4C0099", activebackground="dark red", width=20).place(x=50, y=200 + i *
50, anchor="w")

    def load(self):
        self.rep = filedialog.askopenfilenames()
        if not self.rep:
            return
        img = cv2.imread(self.rep[0])
        if img is None:
            messagebox.showerror("Error", "Could not load image.")
            return
        self.display_image(img, self.image_labels[0])

    def display_image(self, img, label):
        resized_img = cv2.resize(img, (200, 200))
        self.from_array = Image.fromarray(resized_img)
        render = ImageTk.PhotoImage(self.from_array)
        label.config(image=render)
        label.image = render

    def preprocessing(self):
        img = cv2.imread(self.rep[0])
        hsv_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        self.display_image(hsv_img, self.image_labels[1])

    def segmentation(self):
        img = cv2.imread(self.rep[0])
        hsv_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

```
        gauss = cv2.GaussianBlur(hsv_img, (5, 5), 0)

        pixel_values = gauss.reshape((-1, 1))
        pixel_values = np.float32(pixel_values)
        criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 100,
0.2)
        k = 4
        _, labels, centers = cv2.kmeans(pixel_values, k, None, criteria, 10,
cv2.KMEANS_RANDOM_CENTERS)
        centers = np.uint8(centers)
        segmented_image = centers[labels.flatten()].reshape(gauss.shape)

        self.display_image(segmented_image, self.image_labels[2])

        kernel = np.ones((5, 5), np.uint8)
        dilated_image = cv2.dilate(segmented_image, kernel, iterations=1)
        self.display_image(dilated_image, self.image_labels[3])

    def extract_features(self):
        gray_image = self.from_array.convert('L') if self.from_array.mode !=
'L' else self.from_array
        gray_image = np.array(gray_image)

        distances = [1, 2]
        angles = [0, np.pi / 4, np.pi / 2, 3 * np.pi / 4]
        glcm = graycomatrix(gray_image, distances=distances, angles=angles,
levels=256, symmetric=True, normed=True)

        features = {
            'Contrast': graycoprops(glcm, 'contrast'),
            'Correlation': graycoprops(glcm, 'correlation'),
            'Energy': graycoprops(glcm, 'energy'),
            'Homogeneity': graycoprops(glcm, 'homogeneity')
        }

        self.result_text1 = Text(self, height=15, width=60,
background="white")
        self.result_text1.place(x=550, y=300)
        self.result_text1.delete(1.0, END)
        self.result_text1.insert(END, "\n".join([f"{name}: {value}" for name,
value in features.items()]))

    def classification(self):
        if self.rep is None:
            messagebox.showwarning("Warning", "No image loaded for
classification.")
            return

        test_tensors1 = self.prepare_image_for_prediction(self.rep[0])
        pred1 = self.model.predict(test_tensors1)
```

```
        predicted_class_index1 = np.argmax(pred1)
        self.display_prediction(predicted_class_index1)

    def prepare_image_for_prediction(self, img_path):
        img = load_img(img_path, target_size=(224, 224))
        x = img_to_array(img)
        return np.expand_dims(x, axis=0) / 255

    def display_prediction(self, predicted_class_index):
        class_names = [item[10:-1] for item in sorted(glob("./dataset/*/"))]
        result = f'Predicted disease is:
{class_names[predicted_class_index]}'
        messagebox.showinfo("Prediction Result", result)

    def refresh(self):
        for label in self.image_labels:
            label.config(image='')
            label.image = None
        self.result_text1.delete(1.0, END)
        self.rep = None


class LoginWindow(Frame):
    def __init__(self, master=None):
        super().__init__(master)
        self.master = master
        self.config(bg='white')
        self.master.title("Login")

        bg_image = Image.open("rose.jpg")
        bg_render = ImageTk.PhotoImage(bg_image)
        self.background_label = Label(self, image=bg_render)
        self.background_label.image = bg_render
        self.background_label.place(x=0, y=0, relwidth=1, relheight=1)
        self.pack(fill=BOTH, expand=1)

        text_label = Label(self, text="Welcome to the Roseplant Disease
Prediction", bg='white', fg='black', font=("Helvetica", 16))
        text_label.place(x=450, y=200)

        self.login_button = Button(self, text="upload", command=self.login)
        self.login_button.place(x=650, y=300)

    def login(self):
        self.master.switch_frame(Window)


class MainApplication(Tk):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
```

```
        self.title("Roseplant Disease Prediction")
        self.geometry("1400x720")
        self.current_frame = None
        self.switch_frame(LoginWindow)

    def switch_frame(self, frame_class):
        new_frame = frame_class(self)
        if self.current_frame:
            self.current_frame.destroy()
        self.current_frame = new_frame
        self.current_frame.pack()


if __name__ == "__main__":
    app = MainApplication()
    app.mainloop()
```