A Project Report on

Leaf Identification and Disease Diagnosis

Submitted in partial fulfillment for award of

Bachelor of Technology

Degree

in

Data Science

By

K.Durga Bhavani (Y20ADS411) **V.Mani Shankar(Y20ADS429)**

P.Chandu (Y20ADS419) **P.Komalesh (Y20ADS422)**

Under the guidance of

M. Babu Rao, M. Tech

Asst. Professor.

Department of Cyber Security, Data Science & AIML

Bapatla Engineering College

(Autonomous)

(Affiliated to Acharya Nagarjuna University)

BAPATLA – 522 102, Andhra Pradesh, INDIA

2023-2024

Department of
Cyber Security, Data Science & AIML

CERTIFICATE

This is to certify that the project report entitled **Leaf Identification and Disease Diagnosis** that is being submitted by K. Durga Bhavani (Y20ADS411),V.ManiShankar (Y20ADS429),P.Chandu (Y20ADS419), P. Komalesh (Y20ADS422) in partial fulfillment for the award of the Degree of Bachelor of Technology in Data Science to the Acharya Nagarjuna University is a record of bonafide work carried out by them under our guidance and supervision.

Signature of the Guide	**Signature of the HOD**
M. Babu Rao, M. Tech	**Prof. V. Chakradhar, M.Tech**
Assistant Professor	**Hod of CB, DS & AIML**

Signature of the External Examiner

DECLARATION

We declare that this project work is composed by ourselves, that the work contained herein is our own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

K. Durga Bhavani	(Y20ADS411)
V. Mani Shankar	(Y20ADS429)
P. Chandu	(Y20ADS419)
P. Komalesh	(Y20ADS422)

ACKNOWLEDGEMENT

We sincerely thank the following distinguished personalities who have given their advice and support for successful completion of the work.

We are deeply indebted to our most respected guide **M. BABU RAO,** Asst. Prof., Department of CB, DS & AIML, for his valuable and inspiring guidance, comments, suggestions, and encouragement.

We extend our sincere thanks to **V. CHAKRADHAR**, Prof. & Head of the Dept. of CB, DS & AIML, for extending his cooperation and providing the required resources.

We would like to thank our beloved Principal, **Dr. NAZEER SHAIK,** for providing the online resources and other facilities to carry out this work.

We would like to express our sincere thanks to our project coordinator **G. V. LEELA KUMARI,** Asst. Prof. Dept. of CB, DS & AIML for her helpful suggestions in presenting this document.

We extend our sincere thanks to all other teaching faculty and non-teaching staff of the department, who helped directly or indirectly for their cooperation and encouragement.

K. Durga Bhavani (Y20ADS411)

V. Mani Shankar (Y20ADS429)

P. Chandu (Y20ADS419)

P. Komalesh (Y20ADS422)

TABLE OF CONTENTS

Contents **Page.No**

LIST OF FIGURES viii

LIST OF TABLES x

ABSTRACT xi

CHAPTER – 1 1

1 INTRODUCTION 1

1.1 Introduction 1

1.2 Motivation 2

1.3 Objective 2

CHAPTER – 2 3

2 LITERATURE SURVEY 3

CHAPTER – 3 7

3 PROBLEM STATEMENT 7

CHAPTER – 4 8

4 SYSTEM ANALYSIS 8

4.1 Existing System 8

4.1.1 Result and Analysis of Existing System 9

4.1.2 Disadvantages of Existing System 9

4.2 Proposed System 10
4.2.1 Advantages of Proposed System 10
CHAPTER – 5 11
5 SYSTEM REQUIREMENTS 11
5.1 Software Requirements 11
5.2 Hardware Requirements 12
5.3 Libraries Used 12
CHAPTER – 6 16
6 SYSTEM DESIGN 16
6.1 System Architecture for Proposed System 16
6.2 Database Creation 17
6.3 UML Diagrams 18
6.3.1 Use Case Diagram 19
6.3.2 Class Diagram 20
6.3.3 Sequence Diagram 21
6.3.4 Activity Diagram 22
CHAPTER – 7 23
7 IMPLEMENTATION 23
7.1 Dataset 23
7.2 Pre-processing of plant village dataset 26
7.3 Algorithms Used 27

7.3.1 Random Forest Algorithm 27

7.3.2 Convolutional Neural Networks 32

7.4 Feature Extraction 37

7.5 Classification and Prediction 39

7.6 Flask application development 39

7.7 Web application 41

7.8 Deployment 41

7.9 Source Code Screen Shots 42

7.9.1 Leaf Identification Code 42

7.9.2 Disease Prediction Code 43

CHAPTER – 8 47

8 TESTING 47

8.1 Tests Used In the Project 47

8.2 Testing Techniques 48

8.2.1 Black-Box testing 48

8.2.2 White-Box testing 49

8.3 Test Cases 49

CHAPTER – 9 51

9 RESULTS 51

9.1 Screen Shots of Dataset 51

9.2 Identification of leaf and disease prediction 53

CONCLUSION..55

FUTURE ENHANCEMENT..56

REFERENCES 57

LIST OF FIGURES

Name of the Figure **Page.No**

Figure 6.1 System Design for Leaf Identification & Disease Diagnosis 16

Figure 6.2 Image Database 18

Figure 6.3 Use Case Diagram for work of user interface 19

Figure 6.4 Class Diagram for model building 20

Figure 6.5 Sequence Diagram for getting status of the leaf image 21

Figure 6.6 Activity Diagram for flow of work 22

Figure 7.1 Sample Images of Plant Village Dataset 25

Figure 7.2 Random Forest Algorithm 28

Figure 7.3 Convolutional Neural Networks 33

Figure 7.4 Extracting the Features from Leaf Image 38

Figure 7.5 Function to iterate over the leaf image folders 42

Figure 7.6 Training to the model and Accuracy 43

Figure 7.7 Convolutional Neural Networks Model building 43

Figure 7.8 Convolutional Neural Network model Summary 44

Figure 7.9 Training to Epochs 45

Figure 7.10 Performance Evaluation 45

Figure 9.1 Plant Village Dataset from 1 to 19 species..51

Figure 9.2 Plant Village Dataset from 20 to 38 species..52

Figure 9.3 Plant Name & Disease Prediction for Apple Leaf.......................................53

Figure 9.4 Plant Name & Disease Prediction for Cherry Leaf 54

LIST OF TABLES

Name of the Table **Page.No**

Table 4-1 Accuracy and Loss for Existing System 9

Table 7-1 Accuracy and Loss for our Model 46

Table 8-1 Test Cases 50

ABSTRACT

In the absence of plants we would not be able to live on this earth as agriculture plays a vital role in countries growth .One of the most key aspect in agriculture field is plant disease identification that will reduces the crop quality and quantity .The physical recognition of leaf disease in the plant is more time consuming as we need experts. Early and accurate identification of plant diseases can help prevent the spread of infections and reduce the losses.

In this project, we propose a novel method for plant leaf image identification and its disease recognition using **Random Forest algorithm and Convolutional Neural Network (CNN)** in machine learning model. We use CNN for recognition and then use Random forest algorithm for plant diseases prediction. This identification is essential for various areas like ecology, horticulture, disease detection and conservation. Our method can provide a fast and reliable solution for plant and its disease identification using machine learning algorithm for satisfactory results with high accuracy.

KEY WORDS: Random Forest, Convolutional Neural Networks, Ecology, Agriculture.

CHAPTER – 1

1 INTRODUCTION

1.1 Introduction

Plants are like the superheroes of our planet, quietly working behind the scenes to keep everything running smoothly. They provide the air we breathe, the food we eat, and even help stabilize our climate by soaking up carbon dioxide. Think of them as the ultimate caretakers of our environment, providing homes and food for animals and keeping ecosystems balanced.

But plants aren't just important for nature—they're also vital for our economies and cultures. Farmers rely on plants to grow crops that feed us and support livelihoods. Plus, plants have been part of human history and traditions for ages, inspiring everything from art to medicine. So, when we talk about the importance of plants, we're really talking about the foundation of life itself—the incredible web of connections that keeps our world spinning.

In the realm of computer vision, image processing and classification algorithms play a pivotal role in extracting meaningful information from visual data. These algorithms enable machines to interpret and understand images, opening doors to a wide range of applications, from medical diagnostics to autonomous vehicles. Image processing encompasses a broad spectrum of techniques aimed at manipulating and enhancing digital images to extract useful information or improve visual quality. These techniques may include filtering, edge detection, noise reduction, and feature extraction. By pre-processing images before analysis, image processing algorithms help optimize the performance of subsequent classification models.

1.2 Motivation

The main motivation behind this project is to help the agriculture farming and the people who are maintaing home gardens and in the fields like horticulture, ecology etc.So our main moto is to help to the people who are doing farming inorder to provide food to the growing population and to increase the crop yield for the farmers and also for the small gardens.

1.3 Objective

The objective of this project is to develop a robust and accurate system for plant leaf identification and disease classification. By leveraging the power of machine learning and deep learning techniques, our aim is to create a solution that can accurately identify different plant species based on their leaf characteristics and detect diseases affecting these plants. The key objectives of the project include Plant Leaf Identification, Disease Classification, Integration and Deployment.

CHAPTER – 2

2 LITERATURE SURVEY

1. Plant Identification With Convolutional Neural Networks [1].

Publication Year: 2015

Author: Sue Han Lee, Chee Seng Chan, Paul Wilkiny,Paolo Remagnino

Journal Name: IEEE International Conference.

Summary: In the active work to study neural network, we study 44 distinctive plant species, to administer learning characteristics from Royal Botanic Gardens, England, we have undertaken 2816 images in a dataset. Neural networks' de-convolution has been additionally been completed all together to think about the characteristics. The Convolutional Neural Networks approach is aimed to improve the classification of species by leveraging deep learning techniques. Notably, they found that certain features, were uniquely representative of each plant species. The study demonstrated the superiority of the CNN approach over traditional methods.

Advantage: The paper employs Convolutional Neural Networks, which are powerful deep learning models for image analysis and can generalize well to unseen plant species and can handle complex patterns.

Disadvantage: Convolutional Neural networks includes large amount of labelled data for effective training and are prone to overfitting if not properly regularized. Challenges include data requirements, computational complexity, interpretability etc.

2. Analysis Of Transfer Learning For Deep Neural Network Based Plant Classification Models [2].

Publication Year: 2019

Author: Kaya, Keceli, Catal, Yalic, H.Y., Temucin, H.And Tekinerdogan.

Journal Name: Computers And Electronics In Agriculture 158 (2019).

Summary: The confine understanding for classification is contemplated by this paper, Which reproduce the results of the 4 diver transfer literacy model. Collecting of DNN grouped plant we require four distinctive public datasets. Eventually, it has been shown by their trial research that Transfer Learning gives a self-assessing as well as examineplant grouping model. The modeling of a plant is examined by flatten the input image dimension to 1D, normalize the image pixel value, one hot encode column, build a model architecture. Manual plant species classification is time consuming and requires expert knowledge. To address this, machine learning algorithms, including DNNs, have been proposed which will improve the automated plant identification by leveraging transfer learning techniques, which involves in reusing knowledge gained from different applications of DNNs

Advantage: Transfer learning allows leveraging knowledge from pre-trained models, significantly reducing time required to train, leads to better neural network performance and fine tuning, etc.

Disadvantage: Transfer learning assumes that the source and target are related. If the domains differ significantly, negative transfer can occur and pre-trained model may lead to overfitting if the dataset is too small or noisy.

3. Factors influencing the use of deep learning for plant disease recognition [3].

Publication Year: 2018

Author: R. Barbedo, J.G.

Journal Name: Biosystems engineering, 172 (2018).

Summary: Here, probe the basic attorney affecting composition and layout of neural network applied in the pathology of plants it inspects in brief style further. They additionally utilized an exchange-taking in procedure from the recently prepared CNN Model. CNN plant infection acknowledgment is influenced by factors' rundown (that is) Image Captured Conditions, Image Background, Covariate Shift, and Limited Annotated Datasets. Disorders with Similar Symptoms, Simultaneous Disorders, Symptom Variation, Indications Segmentation, are seriously associated with the issue. Traditional visual crop inspection by humans is subject to bias and error. Trained plant pathologists are not always available, especially in remote areas. Automated tools based on images can play a crucial role in detecting and recognizing diseases or patterns. The authors have used image database which is thoroughly analyzed.

Advantage: The paper investigates the use of deep neural networks(DNNs) for plant disease recognition. DNNs can automatically learn relevant features from images, leading to more accurate disease detection. The authors utilize an exchange-learning procedure from pre-trained CNN model of transfer learning.

Disadvantage: Deep learning models require annotated datasets for training. Limited availability of labelled plant images and DNNs can be complex, making it challenging to understand their decision-making process.

4. How deep learning extracts and learns leaf features for plant classification [4].

Publication Year: 2017

Author: Lee, S.H., Chan, C.S., Mayo, S.J. and Remagnino.

Journal Name: Pattern Recognition, 71 (2017).

Summary: This paper articulates a variety of plant leaf pictures alphabetically each plant illustrated is then processed to CNN to detect each plant features. In this, CNN was chiefly utilized for enhanced component portrayal as well as for Leaves' effective discoveries afterword utilized in Deconvolutional Network. That determines of plant's edge by edge. The authors employ deep convolutional feature extraction methods, including Resnet50V2, Inception Resnet V2, MobilenetV2 and VGG16, to extract features from leaf images and learning features directly from the raw input data of hybrid approach which enhance the discriminative power of plant classification systems

Advantage: The paper utilizes CNNs to automatically extract relevant features from raw leaf images data. This eliminates the need for manual feature engineering, which can be time-consuming and subjective.

Disadvantage: Sufficient labelled leaf data for effective training, complex and lack interpretability while feature extraction, choosing right CNN architecture (e.g., Resnet50V2, Inception Resnet V2, MobilenetV2, VGG16) and tuning parameters can impact feature extraction, proper model selection and hyper parameter optimization are essential.

CHAPTER – 3

3 PROBLEM STATEMENT

The problem statement addressed by our project revolves around the need for efficient and accurate identification of plant species and the early detection of diseases affecting these plants. In the context of agriculture and horticulture, timely and precise identification of plants and diseases is crucial for effective crop management, disease control, and maximizing yields. However, manual identification methods are often time-consuming, labor-intensive, and prone to errors, highlighting the necessity for automated solutions.

The main challenges addressed by this project include plant species identification, disease prediction and classification, integration and usability. By resolving these challenges, the project aims to develop a comprehensive solution that enhances agricultural practices, aids in early disease detection, promotes sustainable crop management and ultimately contributes to ensuring food security and agricultural productivity.

CHAPTER – 4

4 SYSTEM ANALYSIS

4.1 Existing System

Identifying plants through their leaves is a thoroughly pursued endeavor that has widely varying applications ranging from ecology, horticulture, disease identification, rare plant preservation in plants to medicinal applications in Ayurveda and various plant bases medical systems. Purpose of this project is to identify plant species digitally using the image of a single leaf through neural networks. We will approach our project using Keras, Tensorflow, and Convolutional Neural Networks. Fore mentioned approach gives satisfactory results with high accuracy.

The dataset comprises 7000 images of seven plants, 1000 from each plant. Data pre-processing includes resizing the image into 180 * 180 pixels, converting the image into an array (using NumPy), and data standardization. Dataset is split into two parts. 80% of specifics are in the training set and 20% of the specifics are in the testing set. Our CNN is sequential, built using layers as follows: Conv2D, MaxPooling2D, Flatten, and Dense. The softmax activation function is utilized on the final thick film to give the chances of individual class.

Sequential CNN is used to train our model rather than parallel CNN because it gives more accurate results. Our image detector is used using Keras, Tensorflow, and CNN. We train the data for 20 epochs for a batch size of 60. NO. oftrainable parameters in our model is 3,989,156. After training, we use 20% of our dataset for testing our model. Pre-stored images are tested after resizing the input images into 180*180 pixels.

4.1.1 Result and Analysis of Existing System

The model was trained for 20 epochs. For training, 5600 images were utilized as well as 1400 images utilized for testing. Our model gives 99.09% accuracy with a loss of 0.00086. A leaf image detector is successfully created using Keras, Tensorflow and CNN. The model will determine the species of given plant with 99% accuracy for prestored images.

Table 4-1 Accuracy and Loss for Existing System

Epoch's Number	Accuracy	Loss	Validation accuracy	Validation Loss
4	96.84	0.0747	98.44	0.0603
8	99.64	0.0109	96.10	0.1560
12	99.78	0.0060	99.22	0.0856
16	99.65	0.0142	89.09	0.4102
20	99.06	0.00086	99.09	0.0596

4.1.2 Disadvantages of Existing System

- Limited number of species with 100 images each.
- Identifies only pre-trained images.
- For interaction with the model, No GUI (like front end application or mobile application) was developed.

4.2 Proposed System

The proposed system integrates Random Forest (RF) and Convolutional Neural Network (CNN) algorithms for plant leaf identification and disease diagnosis. Leveraging RF's ability to handle structured data and CNN's proficiency in image processing, the system offers robust and accurate predictions.

For leaf identification, RF utilizes handcrafted features such as color histograms, texture descriptors, and shape characteristics extracted from leaf images. These features are fed into the RF model, which employs ensemble learning to make predictions based on decision trees collective outputs. Meanwhile, CNN automatically extracts hierarchical features directly from input leaf images, learning patterns and textures through convolutional layers. This end-to-end learning approach enables CNN to capture intricate details and variations in leaf morphology, enhancing classification accuracy.

Furthermore, the system includes a user-friendly web application that allows users to upload leaf images for identification and disease classification. The application provides real-time feedback, displaying the predicted plant species and identifying any detected diseases, aiding in timely plant health management decisions.

4.2.1 Advantages of Proposed System

- Early detection of diseases
- Comprehensive feature extraction
- Scalability
- Adaptability & Transfer Learning

CHAPTER – 5

5 SYSTEM REQUIREMENTS

5.1 Software Requirements

Operating System: The system should support popular operating systems such as Windows, macOS, or Linux. Choose the operating system based on your familiarity, compatibility with required software, and deployment environment preferences.

Python: Install Python, a widely-used programming language for machine learning and data science tasks. Preferably, use the latest stable version of Python (e.g., Python 3.7 or higher) to leverage the latest features and improvements.

Development Environment: Set up a Python development environment with essential tools such as Anaconda, Jupyter Notebook, or any preferred integrated development environment (IDE) like PyCharm or Visual Studio Code. These tools provide a convenient interface for writing, debugging, and executing Python code.

Dependencies: Install necessary Python libraries and dependencies for machine learning and deep learning tasks, including but not limited to scikit-learn, TensorFlow, Keras, NumPy, pandas, and OpenCV. Use package managers like pip or conda to install and manage dependencies effectively.

Web Framework: Flask or Django web framework for building the web application. Flask is lightweight and suitable for smaller projects, while Django offers more features and scalability for larger applications.

Machine Learning Libraries: TensorFlow or PyTorch for implementing machine learning models. These libraries provide tools and APIs for building, training, and deploying deep learning models for image classification and object detection tasks.

5.2 Hardware Requirements

Processor: A multi-core processor with sufficient computational power to handle model training and inference tasks efficiently. A modern CPU with at least quad-core capabilities is recommended.

Memory (RAM): Adequate RAM to accommodate the training and inference processes, especially when working with large datasets and deep learning models. A minimum of 8 GB of RAM is recommended, with higher capacities preferred for more demanding tasks.

Storage: Sufficient storage space to store datasets, trained models, and other related files. SSD storage is preferred over HDD for faster data access speeds, especially during model training and deployment.

Graphics Processing Unit (GPU): Optional but recommended for accelerating deep learning model training and inference. A dedicated GPU with CUDA support, such as NVIDIA GeForce GTX or RTX series, can significantly speed up computation-intensive tasks.

5.3 Libraries Used

numpy (np): NumPy is a fundamental package for scientific computing with Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays efficiently. NumPy is commonly used for array manipulation, mathematical operations, and data pre-processing in machine learning and deep learning workflows.

pandas (pd): Explanation: pandas is a powerful data manipulation and analysis library for Python. It provides data structures like DataFrame and Series, along with functions to manipulate and analyze structured data. pandas is often used for data loading, pre-processing, and manipulation tasks, such as reading data from CSV files, handling missing values, and performing exploratory data analysis.

tensorflow (tf): TensorFlow is an open-source machine learning framework developed by Google. It provides a comprehensive ecosystem of tools, libraries, and resources for building and deploying machine learning models, including deep neural networks. TensorFlow is used for building, training, and evaluating deep learning models, as well as for tasks like data pre-processing, model optimization, and deployment.

hog: The hog function from scikit-image is used for computing Histogram of Oriented Gradients (HOG) features from images. HOG features are commonly used in object detection and image classification tasks. In the code snippet, hog is likely used to extract HOG features from input images as part of the feature extraction process for training the Random Forest classifier.

os: The os module provides a way to interact with the operating system. It is used for tasks such as file and directory manipulation, environment variables, and process management.In the context of the code snippet, the os module may be used for tasks such as navigating directories, checking file existence, or creating directories for storing data or model files.

cv2 (OpenCV): OpenCV (Open Source Computer Vision Library) is a popular open-source computer vision and image processing library. It provides a wide range of functions for tasks such as image reading, manipulation, feature extraction, object detection, and more. In the code snippet, the cv2 module is likely used for image reading (e.g., imread), image resizing, and potentially other image processing tasks such as color conversion, filtering, or edge detection.

Keras: Keras is a high-level neural networks API, capable of running on top of TensorFlow, The key modules imported from Keras include Sequential for creating sequential neural network models, Conv2D and MaxPooling2D for defining convolutional and max pooling layers, Flatten for flattening input data, Dense for adding fully connected layers, and ImageDataGenerator for real-time data augmentation and pre-processing of image data.

Scikit-learn: Scikit-learn is a popular machine learning library in Python that offers a wide range of tools for data mining and analysis. The key modules imported from scikit-learn include RandomForestClassifier for building Random Forest models, train_test_split for splitting datasets into training and testing sets, accuracy_score for computing classification accuracy, and classification_report for generating comprehensive evaluation reports including precision, recall, and F1-score.

Flask: Flask is a lightweight and flexible web framework for Python, known for its simplicity and ease of use in building web applications. It provides tools for routing HTTP requests, rendering HTML templates, and handling form submissions. With Flask, developers can quickly create web applications, APIs, and microservices with minimal boilerplate code. Its modular design and extensive ecosystem of extensions make it suitable for projects of all sizes and complexity levels.

Joblib: Joblib is a Python library for lightweight pipelining in Python. It provides tools for saving and loading Python objects to and from disk. It is commonly used for caching function results, saving machine learning models, and serializing Python objects.

load_model: This function is used to load a pre-trained Keras model from a file. It allows you to load a saved model and use it for inference without having to retrain the model.

Scikit-image: Scikit-image is a collection of algorithms for image processing in Python. It provides tools for image segmentation, feature extraction, image filtering, and geometric transformations.

resize: The resize function in scikit-image is used to resize images to a specified size. It allows you to scale images up or down while preserving aspect ratio and image quality.

image: The image module in Keras.preprocessing provides utilities for loading, pre-processing, and augmenting image data. It includes functions for loading images from files, resizing images, and converting images to arrays for model input.

render_template: This function is used to render HTML templates. It enables you to generate dynamic HTML content by combining static HTML files with data passed from the Python code.

CHAPTER – 6

6 SYSTEM DESIGN

6.1 System Architecture for Proposed System

This machine learning application involves a process of selecting an image from the system and then undergo for a preprocessing. Later it will be assigned to the trained model of both Random Forest and then Convolutional Neural Networks, which will return their result based on the pre trained knowledge they have and the returned values will be caught to the functions written in javascript which will get display on the browser itself.

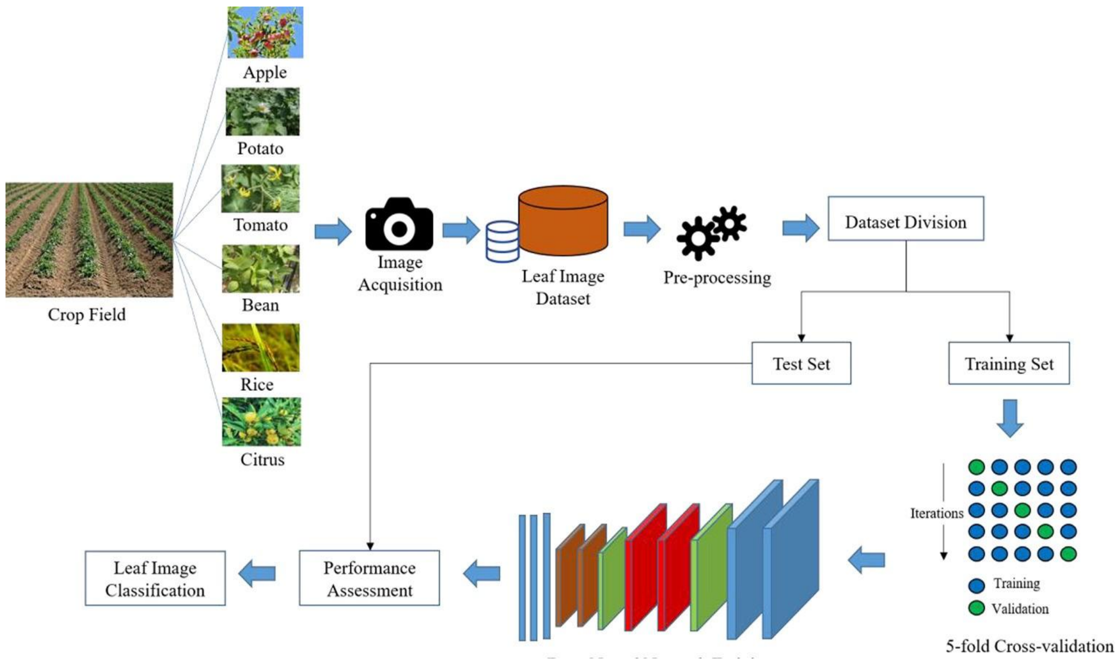


Figure 6.1 System Design for Leaf Identification & Disease Diagnosis

Architectural design: The architectural design of a system emphasizes the design of the system architecture that describes the structure, behavior and more views of that system and analysis.

Logical design: The logical design of a system pertains to an abstract representation of the data flows, inputs and outputs of the system. This is often conducted via modelling, using an over-abstract (and sometimes graphical) model of the actual system. In the context of systems, designs are included. Logical design includes entity-relationship diagrams (ER diagrams).

Physical design: The physical design relates to the actual input and output processes of the system. This is explained in terms of how data is input into a system, how it is verified or authenticated, how it is processed, and how it is displayed. In physical design, the following requirements about the system are decided.

1. Input requirement,
2. Output requirements,
3. Storage requirements,
4. Processing requirements,
5. System control and backup or recovery.

6.2 Database Creation

Database is the organization of data according to a database model. The designer determines what data must be stored and how the data elements interrelate.

With this information, they can begin to fit the data to the database model. Database management system manages the data accordingly.

We will maintain a database where every image that was uploaded in our web application will be saved and stored in the folder. And this data will be helpful to train the model with unseen and new images which will then improve the performance and model uniqueness. In which image classification algorithms will recognise the patterns from the images.

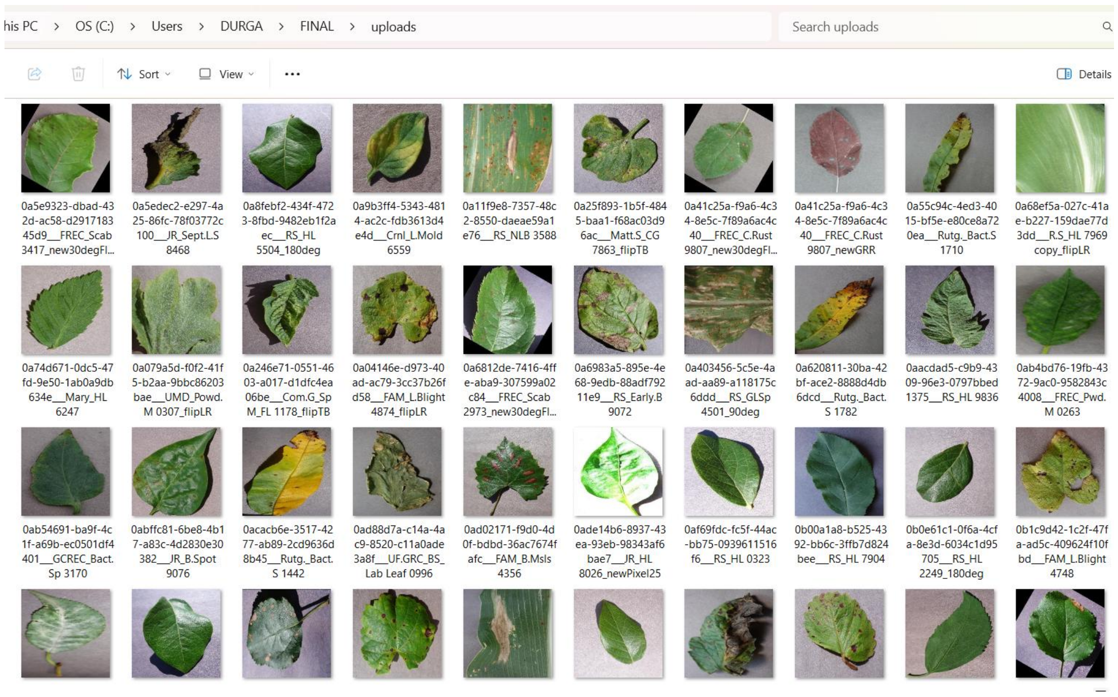

his PC > OS (C:) > Users > DURGA > FINAL > uploads
Search uploads
Sort
View
Details
0a5e9323-dbad-432d-ac58-d291718345d9___FREC_Scab 3417_new30degFl...
0a5edec2-e297-4a25-86fc-78f03772c100___JR_Sept.L.S 8468
0a8febf2-434f-4723-8fbd-9482eb1f2aec___RS_HL 5504_180deg
0a9b3ff4-5343-4814-ac2c-fdb3613d4e4d___Crnl_L.Mold 6559
0a11f9e8-7357-48c2-8550-daeae59a1e76___RS_NLB 3588
0a25f893-1b5f-4845-baa1-f68ac03d96ac___Matt.S_CG 7863_flipTB
0a41c25a-f9a6-4c34-8e5c-7f89a6ac4c40___FREC_C.Rust 9807_new30degFl...
0a41c25a-f9a6-4c34-8e5c-7f89a6ac4c40___FREC_C.Rust 9807_newGRR
0a55c94c-4ed3-4015-bf5e-e80ce8a720ea___Rutg._Bact.S 1710
0a68ef5a-027c-41ae-b227-159dae77d3dd___R.S_HL 7969 copy_flipLR
0a74d671-0dc5-47fd-9e50-1ab0a9db634e___Mary_HL 6247
0a079a5d-f0f2-41f5-b2aa-9bbc86203bae___UMD_Powd.M 0307_flipLR
0a246e71-0551-4603-a017-d1dfc4ea06be___Com.G_SpM_FL 1178_flipTB
0a04146e-d973-40ad-ac79-3cc37b26fd58___FAM_L.Blight 4874_flipLR
0a6812de-7416-4ffe-aba9-307599a02c84___FREC_Scab 2973_new30degFl...
0a6983a5-895e-4e68-9edb-88adf79211e9___RS_Early.B 9072
0a403456-5c5e-4aad-aa89-a118175c6ddd___RS_GLSp 4501_90deg
0a620811-30ba-42bf-ace2-8888d4db6dcd___Rutg._Bact.S 1782
0aacdad5-c9b9-4309-96e3-0797bbed1375___RS_HL 9836
0ab4bd76-19fb-4372-9ac0-9582843c4008___FREC_Pwd.M 0263
0ab54691-ba9f-4c1f-a69b-ec0501df4401___GCREC_Bact.Sp 3170
0abffc81-6be8-4b17-a83c-4d2830e30382___JR_B.Spot 9076
0acacb6e-3517-4277-ab89-2cd9636d8b45___Rutg._Bact.S 1442
0ad88d7a-c14a-4ac9-8520-c11a0ade3a8f___UF.GRC_BS_Lab Leaf 0996
0ad02171-f9d0-4d0f-bdbd-36ac7674fafc___FAM_B.Msls 4356
0ade14b6-8937-43ea-93eb-98343af6bae7___JR_HL 8026_newPixel25
0af69fdc-fc5f-44ac-bb75-0939611516f6___RS_HL 0323
0b00a1a8-b525-4392-bb6c-3ffb7d824bee___RS_HL 7904
0b0e61c1-0f6a-4cfa-8e3d-6034c1d95705___RS_HL 2249_180deg
0b1c9d42-1c2f-47fa-ad5c-409624f10fbd___FAM_L.Blight 4748

Figure 6.2 Image Database

6.3 UML Diagrams

UML stands for Unified Modeling Language. UML is different from the other common programming languages such as C++, Java, COBOL, etc. UML is a pictorial language used to make software blueprints. UML can be described as a general purpose visual modeling language to visualize, specify, construct, and document software system. Although UML is generally used to model software systems, it is not limited within this boundary. It is also used to model non-software systems as well. Types of UML diagrams we have used are as follows:

6.3.1. Use Case Diagram

6.3.2 Class Diagram

6.3.3 Sequence Diagram

6.3.4 Activity Diagram

6.3.1 Use Case Diagram

A use case describes a sequence of actions that provide something of measurable value to an actor and is drawn as a horizontal ellipse an actor is a person, organization, or external system that plays a role in one or more interactions with your system.

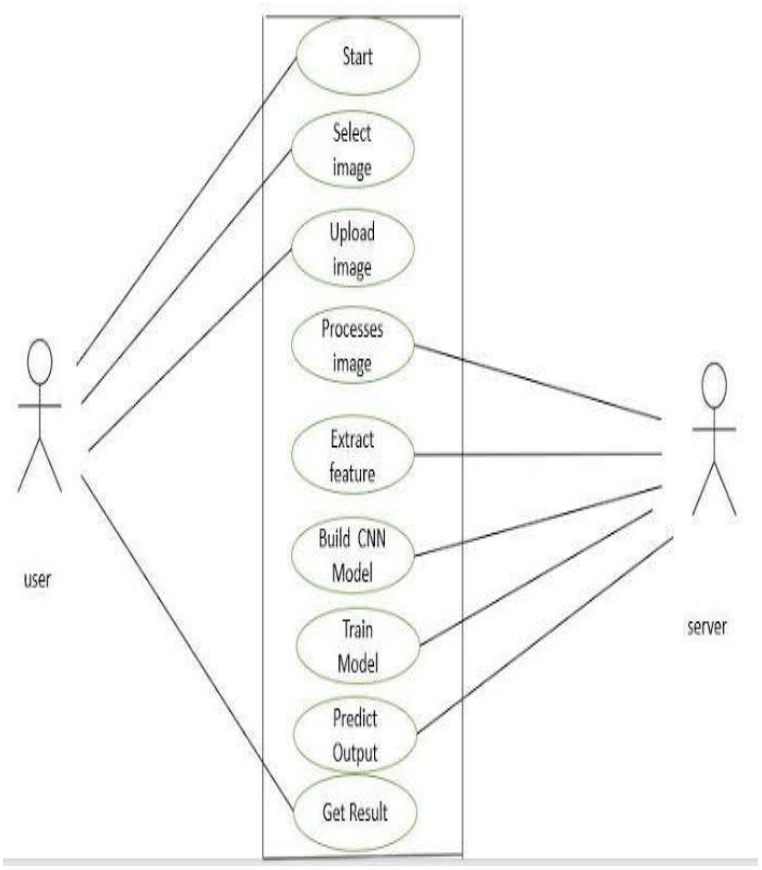


Figure 6.3 Use Case Diagram for work of user interface

6.3.2 Class Diagram

A class diagram describes the static structure of the symbols in your new system. It is a graphic presentation of the static view that shows a collection of declarative (static) model elements, such as classes, types, and their contents and relationships.

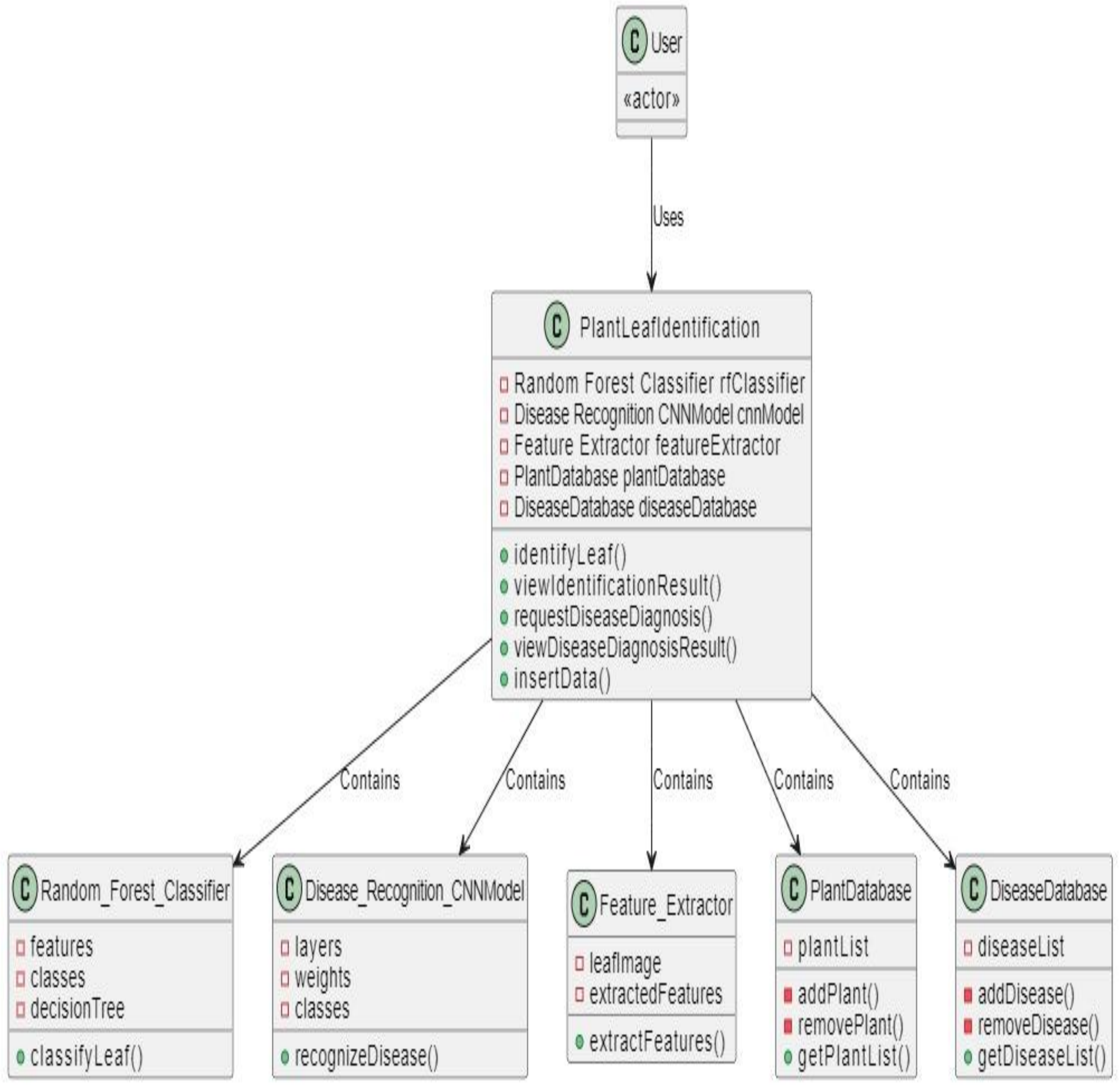


Figure 6.4 Class Diagram model building

6.3.3 Sequence Diagram

UML sequence diagrams model the flow of logic within your system in a visual manner, enabling you both to document and validate your logic, and are commonly used for both analysis and design purposes. Sequence diagrams are the most popular UML artifacts for dynamic modeling, which focuses on identifying the behavior within system.

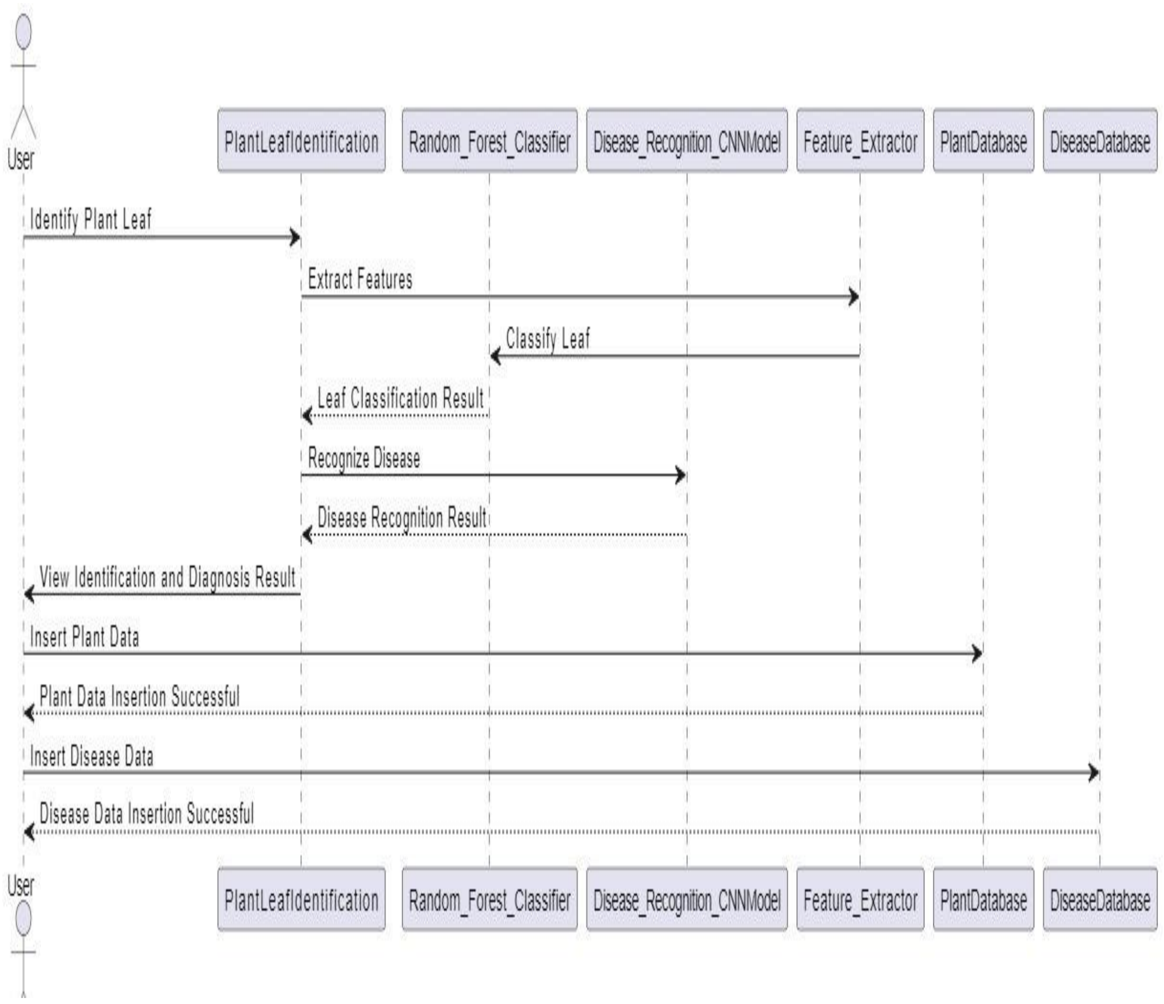


Figure 6.5 Sequence Diagram for getting status of the leaf image

6.3.4 Activity Diagram

Activity diagrams are used to document workflows in a system, from the business level down to the operational level.

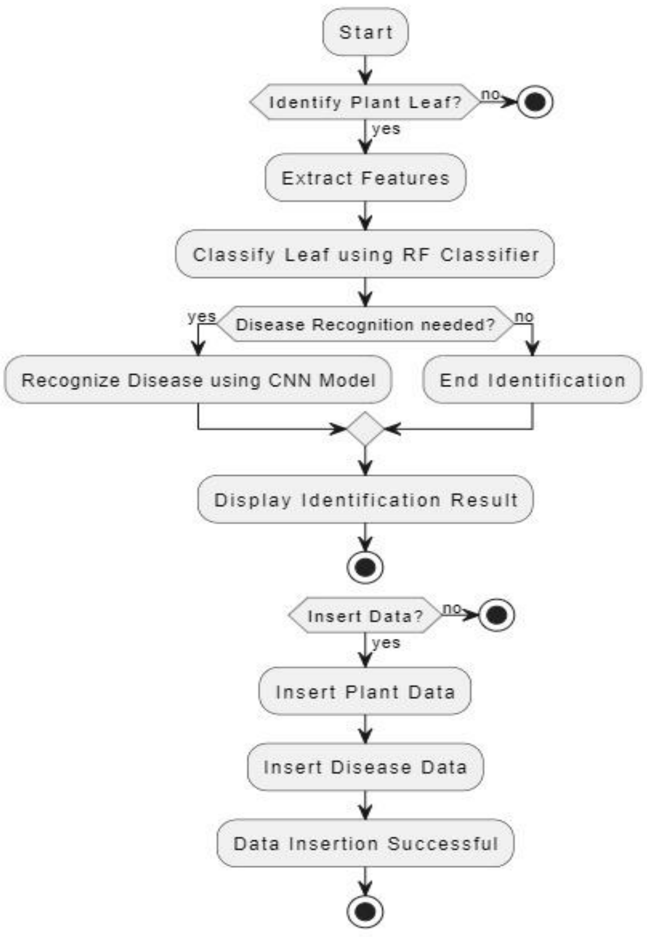

Start
Identify Plant Leaf?
no
yes
Extract Features
Classify Leaf using RF Classifier
Disease Recognition needed?
yes
no
Recognize Disease using CNN Model
End Identification
Display Identification Result
Insert Data?
no
yes
Insert Plant Data
Insert Disease Data
Data Insertion Successful

Figure 6.6 Activity Diagram flow of work

CHAPTER – 7

7 IMPLEMENTATION

7.1 Dataset

The Plant Village dataset is a comprehensive collection of images depicting various plant species along with their associated diseases. It serves as a valuable resource for researchers, developers, and enthusiasts interested in plant pathology, disease detection, and classification using machine learning techniques. Let's delve deeper into this dataset:

The dataset contains high-quality images of diseased and healthy plant leaves, stems, and fruits, captured under controlled conditions to ensure clarity and consistency. Each image is labelled with the plant species, disease type (if applicable), and severity level (if applicable), making it suitable for supervised learning tasks.

Plant Species :

1. Tomato: Tomato plants are one of the primary species featured in the dataset. They are susceptible to various diseases such as Early Blight, Late Blight, and Tomato Yellow Leaf Curl Virus (TYLCV).

2. Potato: Potatoes are another prominent species in the dataset. Common diseases affecting potatoes include Late Blight, Early Blight, and Potato Virus Y (PVY), healthy.

3. Pepper: Peppers, including bell peppers and chilli peppers,are also included. Diseases such as Bacterial Spot and Powdery Mildew are commonly observed.

4. Apple: Apple trees are represented in the dataset, with diseases like Apple Scab and Cedar Apple Rust being significant concern.

5. Blueberry: Blueberry bushes are part of the dataset, with diseases such as Mummy Berry and Blueberry Scorch Virus.

6. Grape: Grapevines are included, with diseases like Downy Mildew and Powdery Mildew being prevalent in vineyards.

7. Peach: Peach trees are represented, with diseases like Peach Leaf Curl and Rot affecting the fruit and foliage.

8. Strawberry: Strawberries are included, with diseases such as Anthracnose and Powdery Mildew being common.

9. Cherry: Typically characterized by its small size, round shape, and red or black. Attributes might include growth habits, preferred climate.

10. Corn: Known for its tall stalks and ears with rows of kernels. Data might cover planting season, soil preferences, water needs, common pests, harvesting techniques, and uses (e.g., animal feed, human consumption).

11. Orange: A citrus fruit tree with round, orange fruit and fragrant blossoms. Data would likely include details on cultivation, pruning, fertilization, irrigation, climate preferences, diseases, and harvesting methods.

12. Raspberry: A shrub with edible, aggregate fruit that typically ranges in color from red to black. Information could encompass planting instructions, spacing requirements, trellising techniques, pest management, pruning schedules, and harvest timing.

13. Soyabean: A legume valued for its high protein content and versatility. Data might cover planting depth, row spacing, nutrient needs (especially nitrogen fixation), weed control methods, diseases like soybean rust, and harvesting practices.

14. Squash: A type of gourd plant producing fruits of various shapes, sizes, and colors. Attributes could include vine training, pollination requirements, soil pH preferences, pest control, disease prevention, and culinary uses for different squash varieties.

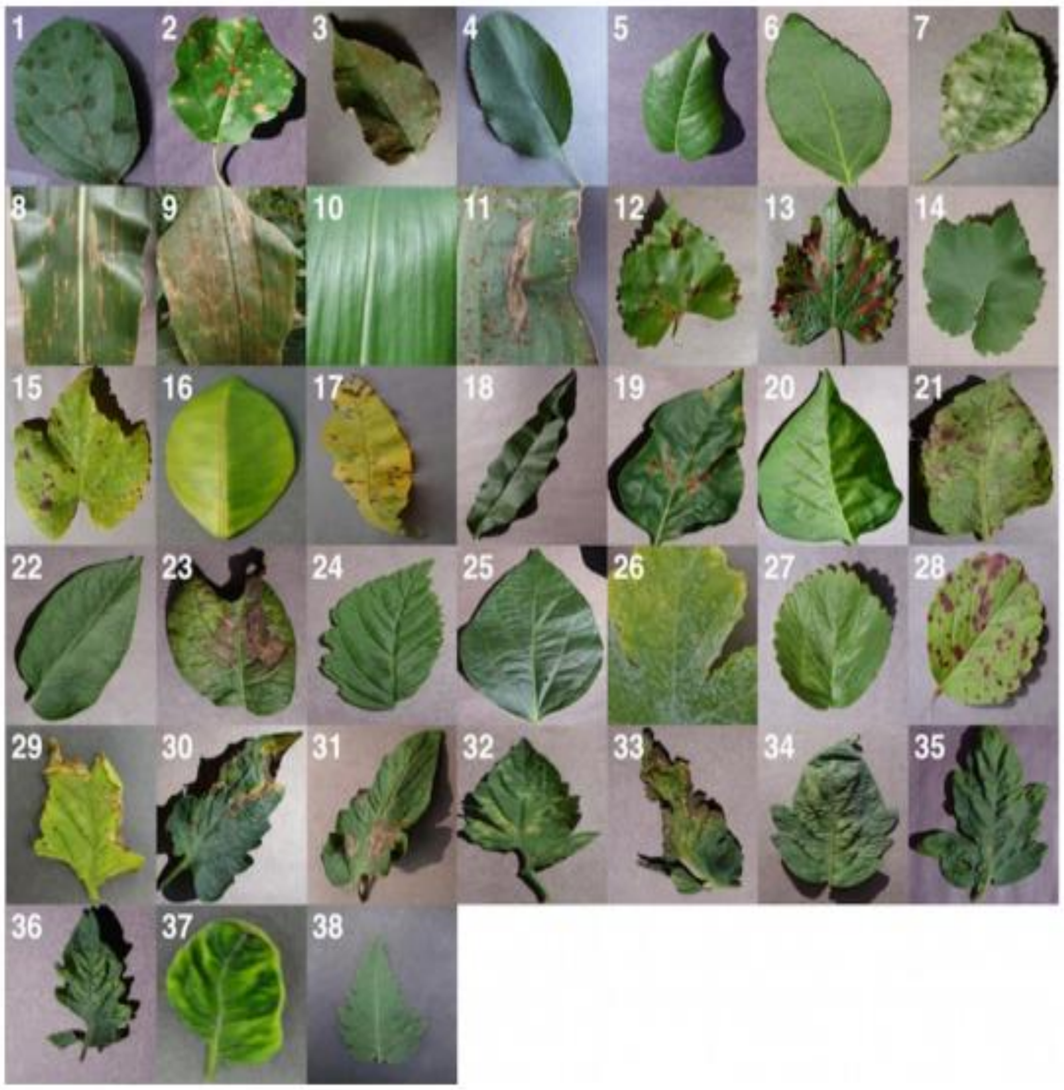


Figure 7.1 Sample Images of Plant Village Dataset

7.2 Pre-processing of plant village dataset

Pre-processing the Plant Village dataset involves several steps to prepare the images and labels for training a machine learning model. Here's a detailed explanation of the pre-processing steps:

1. Data Acquisition: We have downloaded the Plant Village dataset from a reliable source, ensuring that it includes images of plant leaves along with corresponding labels indicating the plant species and disease type.

2. Data Inspection: Before pre-processing, inspect the dataset to understand its structure, including the organization of folders, image file formats, and label annotations. This step helps ensure that the dataset is complete and suitable for the intended analysis.

3. Data Cleaning: Remove any corrupted or incomplete images from the dataset to prevent errors during pre-processing and training. Check for duplicate images and eliminate them to avoid biasing the model towards certain samples.

4. Image Resizing: Resize all images to a uniform size to ensure consistency and compatibility with the model architecture. Commonly used sizes for plant disease datasets range from 224x224 to 512x512 pixels, depending on the specific requirements of the model being used.

5. Normalization: Normalize the pixel values of the images to a common scale, typically ranging from 0 to 1 or -1 to 1. This step helps stabilize training and improves convergence by reducing the effects of variations in pixel intensity across images in the plant village image dataset.

6. Data Augmentation: Augment the dataset by applying transformations such as rotation, flipping, scaling, and cropping to create additional training samples. Data augmentation helps increase the diversity of the training set, making the model more robust to variations in pose, lighting, and background.

7. Label Encoding: Encode categorical labels into numerical format suitable for training machine learning models. For example, convert plant species names and disease types into integer or one-hot encoded representations.

8. Train-Test Split: Split the dataset into training and testing sets to evaluate the model's performance on unseen data. Typically, 70-80% of the data is used for training, while the remaining 20-30% is reserved for testing.

9. Data Loading: Implement a data loader mechanism to efficiently load batches of pre-processed images and labels during model training. Consider using libraries such as TensorFlow Datasets or PyTorch DataLoader for this purpose.

7.3 Algorithms Used

In this project we have used Random Forest algorithm for leaf identification and Convolutional Neural Networks for disease prediction.

7.3.1 Random Forest Algorithm

Random Forest is a popular ensemble learning method used for classification, regression, and other tasks. It belongs to the family of decision tree-based algorithms and is known for its robustness, scalability, and ease of use. RF was introduced by Leo Breiman and Adele Cutler and has since gained widespread popularity in various fields, including machine learning, data science, and bioinformatics.

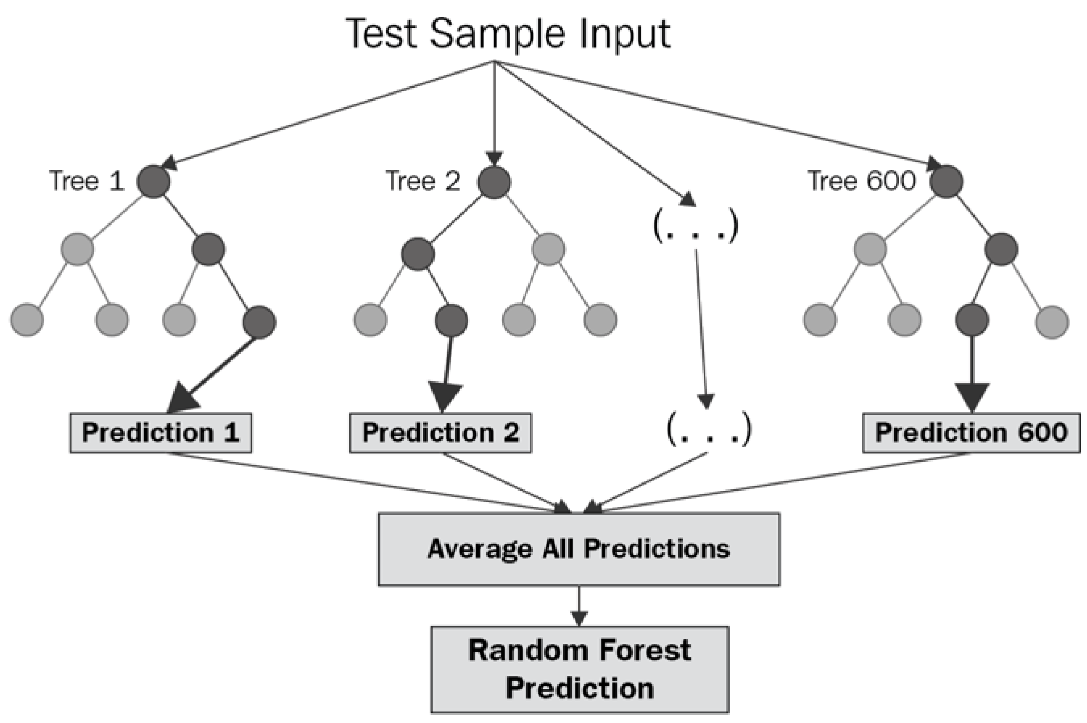


Figure 7.2 Random Forest Algorithm

Key Components of Random Forest:

1. Decision Trees: At the core of Random Forest are decision trees, which are simple yet powerful models for classification and regression. A decision tree recursively splits the input space into regions, making decisions based on feature values to partition the data into classes or predict numerical values. Each internal node of the tree represents a decision based on a feature, while each leaf node corresponds to a class label or a predicted value.

2. Ensemble Learning: Random Forest operates by building an ensemble of decision trees. Instead of relying on a single decision tree, RF trains multiple trees independently on random subsets of the training data and features. This randomness helps reduce overfitting and improves the generalization ability of the model. During inference, predictions from individual trees are aggregated to make the final classification or regression decision.

3. **Bootstrap Aggregating (Bagging):** RF employs a technique called bootstrap aggregating, or bagging, to create diverse training datasets for each decision tree. In bagging, random samples are drawn with replacement from the original training data, resulting in multiple bootstrap samples. Each decision tree is trained on a bootstrap sample, ensuring diversity in the training process and reducing the correlation between trees.

Random Feature Selection: In addition to sampling training data, RF also performs random feature selection at each split of a decision tree. Rather than considering all features when choosing the best split, RF randomly selects a subset of features. This randomness further decorrelates the trees and prevents individual features from dominating the decision-making process.

Advantages of Random Forest:

1. **Robustness to Overfitting:** Random Forests are less prone to overfitting compared to individual decision trees, thanks to the ensemble approach and randomization techniques. This makes RF more resilient to noisy data and outliers.

2. **Scalability:** Random Forests can handle large datasets with high-dimensional feature spaces efficiently due to their parallelizable nature. They can be trained on distributed computing platforms and are suitable for scaling to big data applications.

3. **Feature Importance:** Random Forests provide a measure of feature importance, indicating the contribution of each feature to the model's predictive performance. This information is valuable for feature selection, dimensionality reduction, and understanding the underlying data patterns.

4. **Versatility:** Random Forests can be applied to both classification and regression tasks, making them a versatile algorithm for various machine learning problems. They can handle categorical and numerical data, as well as mixed data types..

5. **Ease of Use:** Random Forests are relatively easy to implement and require minimal hyperparameter tuning compared to other complex algorithms like neural networks. They have fewer parameters to optimize and are less sensitive to parameter choices, making them suitable for practitioners with limited machine learning expertise.

Applications of Random Forest:

1.**Predictive Modeling:** Random Forest is widely used in predictive modeling applications across various industries such as finance, healthcare, and marketing. It can predict outcomes such as customer churn, stock prices, disease diagnosis, and customer preferences based on historical data.

2. **Image Classification:** In computer vision, Random Forests are employed for image classification tasks where the goal is to assign a label to an image based on its content. For example, RF can classify images into different categories such as animals, vehicles, or objects in surveillance systems and automated image tagging applications.

3. **Object Detection:** Random Forests are also used in object detection tasks where the goal is to identify and locate objects within an image. This is commonly used in applications such as autonomous vehicles, facial recognition systems, and industrial inspection for detecting defects in manufactured products.

4. **Remote Sensing and Environmental Monitoring:** In remote sensing applications, Random Forests are utilized to classify land cover types, monitor deforestation, assess vegetation health, and analyze environmental changes over time. RF can process satellite imagery and aerial photographs to provide valuable insights for ecological research and natural resource management.

5. **Bioinformatics and Genomic Analysis:** Random Forests are applied in bioinformatics for tasks such as gene expression analysis, protein structure prediction, and disease diagnosis based on genetic data. RF models can identify patterns in biological datasets, classify gene expression profiles, and predict the likelihood of diseases based on genomic markers.

6. **Anomaly Detection:** Random Forests are effective for anomaly detection in cybersecurity applications, where the goal is to identify unusual or suspicious behavior in network traffic, user activity, or system logs. RF models can detect anomalies by learning normal patterns and flagging deviations from the norm, helping to identify security threats and prevent cyberattacks.

7. Regression: Random Forest can also perform regression tasks, where it predicts continuous numerical values instead of discrete classes. Regression applications include predicting house prices, stock market trends, demand forecasting, and estimating crop yields based on environmental factors.

8. Feature Importance Analysis: Random Forest provides a measure of feature importance, indicating the relative importance of input features in making predictions. This analysis is valuable for understanding the underlying patterns in the data and identifying the most influential factors driving the outcomes.

9. **Imbalanced Data Classification:** Random Forest is robust to imbalanced datasets, where one class is significantly more prevalent than others. It can handle imbalanced data well and is commonly used in applications such as rare disease detection, anomaly detection, and event prediction in security systems.

10. **Ensemble Learning:** Random Forest is a type of ensemble learning method that combines multiple decision trees to improve prediction accuracy and robustness. It mitigates overfitting and reduces variance by averaging predictions from multiple trees, making it suitable for high-dimensional datasets with noisy or missing data.

7.3.2 Convolutional Neural Networks

Convolutional Neural Networks (CNN), a class of deep learning algorithms, revolutionized the field of image classification by introducing hierarchical feature learning. Inspired by the organization of the visual cortex in the human brain, CNNs employ layers of convolutional and pooling operations to automatically learn discriminative features from raw pixel data. Through successive layers, CNNs extract increasingly complex features, enabling them to accurately classify images based on learned patterns. CNNs have demonstrated exceptional performance in tasks such as object detection, facial recognition, and medical image analysis, owing to their ability to capture spatial hierarchies and invariant representations.

Convolutional Neural Networks (CNNs) are a class of deep learning models that have revolutionized the field of computer vision. They are specifically designed to process and analyze visual data such as images and videos, making them highly effective for tasks like image classification, object detection, and image segmentation.

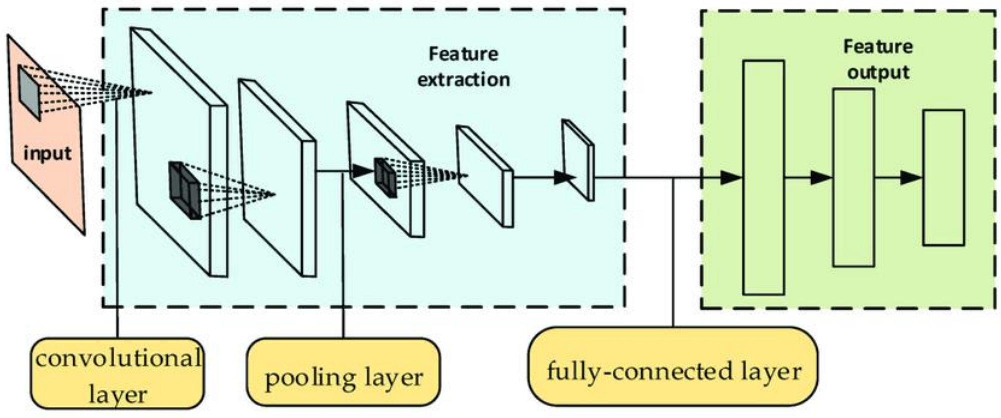


Figure 7.3 Convolutional Neural Networks

Key Components of Convolutional Neural Networks:

1. Convolutional Layers: Convolutional layers are the building blocks of CNNs. They consist of learnable filters or kernels that convolve over the input image to extract features. Each filter detects specific patterns or features, such as edges, textures, or shapes, by performing element-wise multiplications and summations with local patches of the input image. The resulting feature maps capture hierarchical representations of the input data, progressively learning more complex features in deeper layers of the network.

2. Pooling Layers: Pooling layers are used to downsample the spatial dimensions of feature maps, reducing computational complexity and controlling overfitting. Common pooling operations include max pooling and average pooling, which aggregate information from local regions of the feature maps to retain the most relevant features while discarding redundant information. Pooling helps improve translation invariance and reduces the sensitivity of the network to small spatial variations in the input data.

3. Activation Functions: Activation functions introduce nonlinearity into the network, enabling CNNs to learn complex mappings between input and output data. Popular activation functions used in CNNs include ReLU (Rectified Linear Unit), sigmoid, and tanh. ReLU is commonly preferred due to its simplicity and effectiveness in alleviating the vanishing gradient problem, which can hinder the training of deep neural networks.

4. Fully Connected Layers: Fully connected layers, also known as dense layers, are typically used towards the end of the CNN architecture to perform high-level reasoning and decision-making based on the extracted features. These layers connect every neuron in one layer to every neuron in the subsequent layer, allowing the network to learn complex relationships between features and make predictions for classification or regression tasks.

Advantages of Convolutional Neural Networks:

Hierarchical Feature Learning: CNNs automatically learn hierarchical representations of visual data, starting from low-level features such as edges and textures and progressing to high-level semantic features that capture complex object shapes and structures. This hierarchical feature learning enables CNNs to effectively extract discriminative features from images without the need for manual feature engineering.

Translation Invariance: CNNs exhibit translation invariance, meaning they can recognize objects in an image regardless of their position or orientation. This property is achieved through the use of shared weights in convolutional layers, which enable the network to detect features irrespective of their spatial location within the input image.

Parameter Sharing: CNNs leverage parameter sharing across spatial locations, reducing the number of learnable parameters and enhancing the generalization ability of the model. By sharing weights and biases across different regions of the input image, CNNs can effectively capture spatial patterns and variations while maintaining a compact model size.

Scalability: CNNs are highly scalable and can be applied to large-scale datasets and high-dimensional input data, such as high-resolution images and videos. They can be trained on powerful hardware accelerators such as GPUs (Graphics Processing Units) to expedite the training process and handle computationally intensive tasks efficiently.

Transfer Learning: CNNs trained on large-scale image datasets, such as ImageNet, can be fine-tuned or used as feature extractors for downstream tasks with limited labeled data. This transfer learning approach leverages the prelearned representations of features in the early layers of the network, enabling CNNs to achieve superior performance on specific tasks with minimal additional training.

Applications of Convolutional Neural Networks:

Image Classification: CNNs are widely used for image classification tasks, where the goal is to assign a label or category to an input image. They have achieved state-of-the-art performance on benchmark datasets such as ImageNet, surpassing human-level accuracy in some cases.

Object Detection: CNNs are employed for object detection tasks, where the goal is to localize and classify objects within an image. They can detect multiple objects of varying sizes and shapes, making them suitable for applications such as autonomous driving, surveillance, and medical imaging.

Semantic Segmentation: CNNs are used for semantic segmentation, where each pixel in an image is assigned a class label to segment objects and regions of interest. This is valuable for applications such as medical image analysis, scene understanding, and augmented reality.

Face Recognition: CNNs are utilized for face recognition tasks, enabling biometric authentication and identity verification in security systems, mobile devices, and social media platforms.

Medical Imaging: CNNs are applied in medical imaging for tasks such as disease diagnosis, tumor detection, and medical image analysis. They assist healthcare professionals in interpreting medical images accurately and efficiently, leading to improved patient outcomes and diagnosis.

Image Generation: CNNs can generate realistic images by learning to generate new samples from a learned distribution. Applications include generating synthetic images for data augmentation, image inpainting, and creating realistic textures and artwork.

Natural Language Processing (NLP): CNNs can be applied to NLP tasks such as text classification, sentiment analysis, and named entity recognition. They can operate on word embeddings or character-level representations of text to extract features and make predictions.

Video Analysis: CNNs can analyze video data by processing individual frames and capturing temporal dependencies between frames. Applications include action recognition, video surveillance, and video summarization

Biometrics: CNNs are used in biometric authentication systems for tasks such as face recognition, fingerprint recognition, and iris recognition. They can learn discriminative features from biometric data and match them against stored templates for identity verification.

Remote Sensing: CNNs are applied to remote sensing data for tasks such as land cover classification, crop monitoring, disaster management, and environmental monitoring. They can analyze satellite imagery to detect changes in land use, identify vegetation types, and assess environmental conditions.

7.4 Feature Extraction

Feature extraction in the context of the Plant Village dataset involves extracting meaningful features from the images of plant leaves.

Feature Extraction using Random Forest: In RF, feature extraction often involves manually engineering features from input data. For plant leaf identification, these features may include color histograms, texture descriptors (e.g., Haralick features), shape characteristics, and other morphological attributes extracted from the leaf images. Feature selection or dimensionality reduction techniques may be employed to reduce the number of features and eliminate redundant or irrelevant information. This helps improve model efficiency and prevent overfitting.

Random Forest allows flexibility in feature selection and engineering, enabling customization of feature sets based on the specific requirements and characteristics of the input data and target task. Geometric features such as leaf shape, size, and symmetry are important for distinguishing between different plant species.

Feature Extraction using Convolutional Neural Networks: Here, In Convolutional Neural Networks, feature extraction is performed automatically by the convolutional layers of the network. These layers learn hierarchical representations of input data through the application of convolutional filters, which capture low-level to high-level features such as edges, textures, patterns, and object parts. The convolutional filters in CNNs act as feature extractors, detecting local patterns and spatial relationships within the input images. As the input data passes through successive convolutional layers, higher-level features representing more abstract concepts are extracted.

Pooling layers in CNNs downsample feature maps, reducing spatial dimensions and extracting dominant features, aiding in translation invariance and reducing computational complexity. CNN models often leverage transfer learning, where pre-trained networks are fine-tuned or used as feature extractors for target tasks, benefiting from features learned from large-scale image datasets. Features learned by CNNs are hierarchical, with lower layers detecting simple patterns like edges and textures, while higher layers capture more abstract concepts like object parts and shapes.

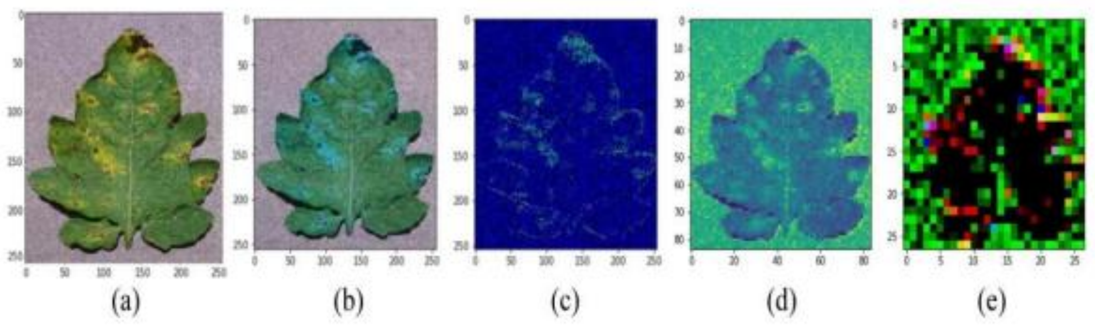


Figure 7.4 Extracting the Features from Leaf Image

7.5 Classification and Prediction

Random Forest employs ensemble learning, where multiple decision trees collectively make predictions based on the extracted features, resulting in robust and accurate classification. In RF classification, predictions are made through a voting scheme, where each decision tree votes for the class label, and the majority class becomes the final prediction. RF provides probabilistic output for classification tasks, indicating the likelihood of each class label based on the aggregated predictions of individual trees. RF prediction is typically fast and efficient, allowing for real-time or near-real-time classification of plant species or disease conditions based on the extracted features.

Convolutional Neural Networks perform end-to-end learning, where both feature extraction and classification are integrated into a single neural network architecture, optimizing performance for the target task. Pre-trained CNN models can be fine-tuned or used as feature extractors for new tasks, leveraging knowledge learned from large-scale datasets to improve classification accuracy. Based on the patterns the model has recognized, will give a prediction of diseased leaf image.

7.6 Flask application development

Here is the process of integrating two machine learning models (for leaf identification and diseases prediction) into a Flask application that takes an image as input and predicts the output.

Load the Models: First, we have loaded two separate machine learning models using a dataset containing images of leaves. One model (Random Forest) is for predicting characteristics of the leaf (e.g., classification leaf type), and the other (Sequential CNN) is for identifying the diseases of the leaf. Each model would have been trained separately and saved as an .h5 file, which is a common format for saving trained Keras models in Python.

Handling Image Uploads: The Flask application would include a route that allows users to upload an image file. This route would handle the uploaded image and pass it to the prediction functions. The uploaded image would typically be stored temporarily on the server or processed directly from memory, depending on your application's requirements. And also the uploaded image will be stored in the database.

Image processing: Once the image is received by the Flask route, you'd pre-process it to ensure it's in the correct format and size expected by the machine learning models. This might involve resizing the image, normalizing pixel values, and converting it to the appropriate data type (e.g., NumPy array) for input into the models.

Model prediction: After pre-processing, we will pass the image data through each of the trained models. Each model will produce predictions based on its respective task. For example, one model might predict disease classification or leaf type, while the other model predicts the species of the leaf. The predictions generated by the models are typically in the form of class labels.

7.7 Web application

When building a web application using Flask, several key components are essential to ensure the application functions correctly and meets the requirements of users. Here are the key components required in a Flask web application

Templates: Templates are HTML files that define the structure and layout of your web pages. Flask uses the Jinja2 templating engine, allowing you to inject dynamic content into your HTML templates. Templates enable you to create dynamic web pages that can display data retrieved from the backend.

Static Files: Static files include CSS style sheets, JavaScript files, images, and other assets that are served directly to the client without any processing by the server. Flask provides a built-in mechanism for serving static files from a directory, typically named static.

Forms: Forms allow users to submit data to the server, such as user input or file uploads. Flask integrates with WTForms, a flexible form validation and rendering library for Python.

7.8 Deployment

Deploying a Flask web application involves several steps to ensure that your application is accessible to users on the internet and can handle production-level traffic securely. Here are the general deployment steps for deploying a Flask application.

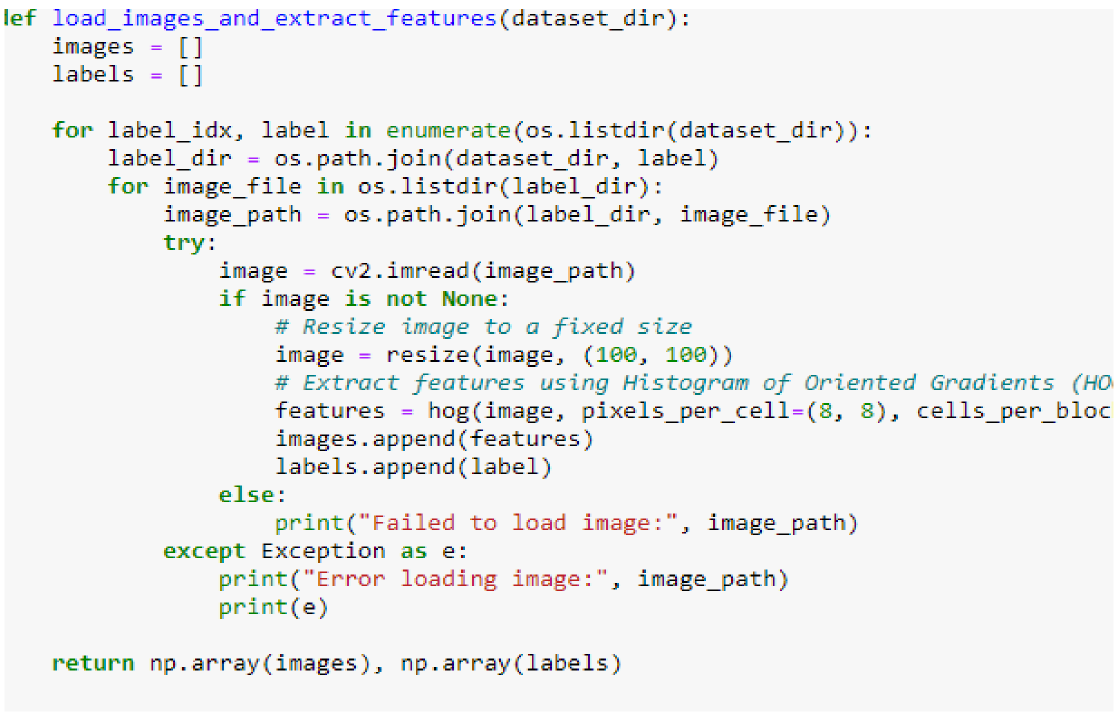
Prepare your Application: we have ensued that the Flask application is fully developed and tested locally before deploying it to a production environment. Make sure all dependencies are documented and included in your project's requirements file (requirements.txt).

Choose Hosting Server: Deploying Flask application to the chosen (chrome browser) hosting provider using the deployment script or manual instructions provided by the provider. Ensure that your application is accessible over the internet and test it to verify that it functions correctly in the production environment.

Monitor of Application: Set up monitoring and alerting to track the health and performance of your deployed Flask application. Monitoring server resources, application logs, error rates, and response times to identify and address any issues promptly. Regularly updating dependencies, apply security patches, and perform backups to ensure the reliability and security of your deployed application.

7.9 Source Code Screen Shots

7.9.1 Leaf Identification Code

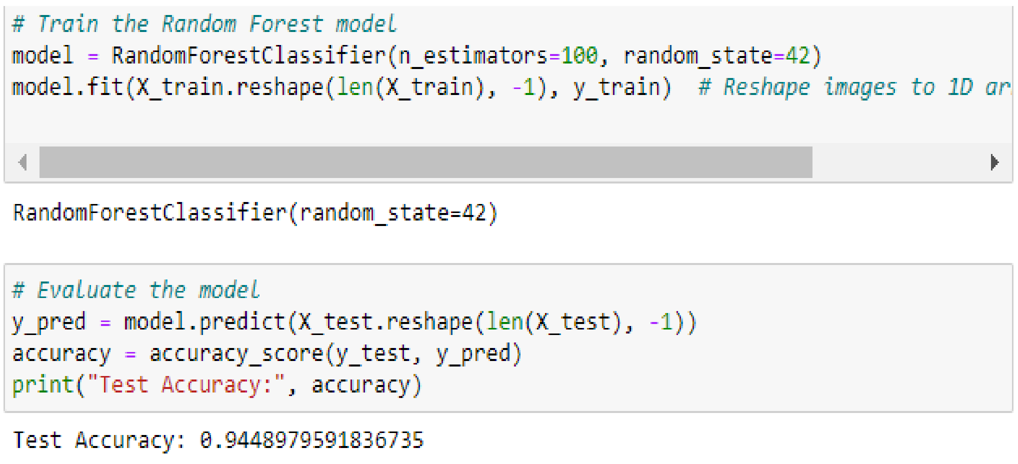


```
lef load_images_and_extract_features(dataset_dir):
    images = []
    labels = []

    for label_idx, label in enumerate(os.listdir(dataset_dir)):
        label_dir = os.path.join(dataset_dir, label)
        for image_file in os.listdir(label_dir):
            image_path = os.path.join(label_dir, image_file)
            try:
                image = cv2.imread(image_path)
                if image is not None:
                    # Resize image to a fixed size
                    image = resize(image, (100, 100))
                    # Extract features using Histogram of Oriented Gradients (HO
                    features = hog(image, pixels_per_cell=(8, 8), cells_per_bloc
                    images.append(features)
                    labels.append(label)
                else:
                    print("Failed to load image:", image_path)
            except Exception as e:
                print("Error loading image:", image_path)
                print(e)

    return np.array(images), np.array(labels)
```

Figure 7.5 Function to iterate over the leaf image folders



```
# Train the Random Forest model
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train.reshape(len(X_train), -1), y_train)  # Reshape images to 1D ar
```

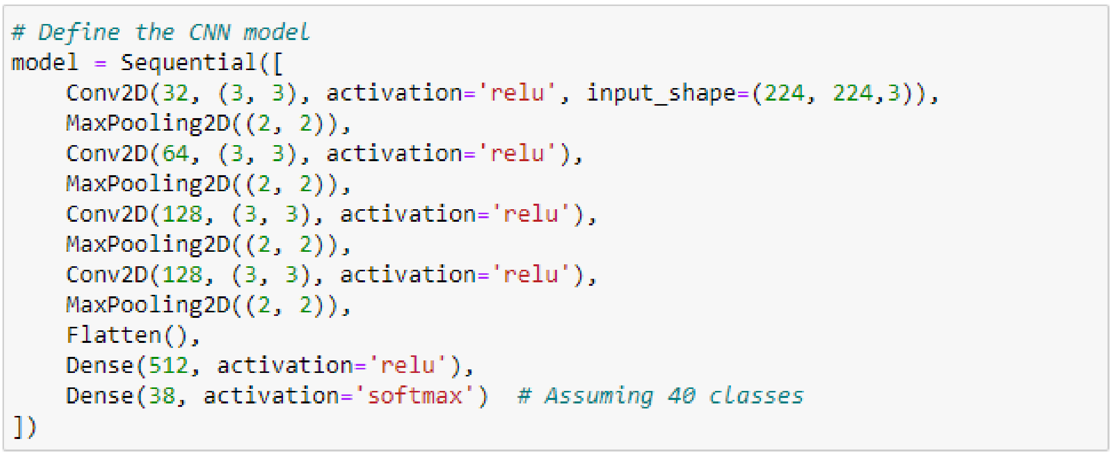
```
RandomForestClassifier(random_state=42)
```

```
# Evaluate the model
y_pred = model.predict(X_test.reshape(len(X_test), -1))
accuracy = accuracy_score(y_test, y_pred)
print("Test Accuracy:", accuracy)
```

```
Test Accuracy: 0.9448979591836735
```

Figure 7.6 Training to the model and Accuracy

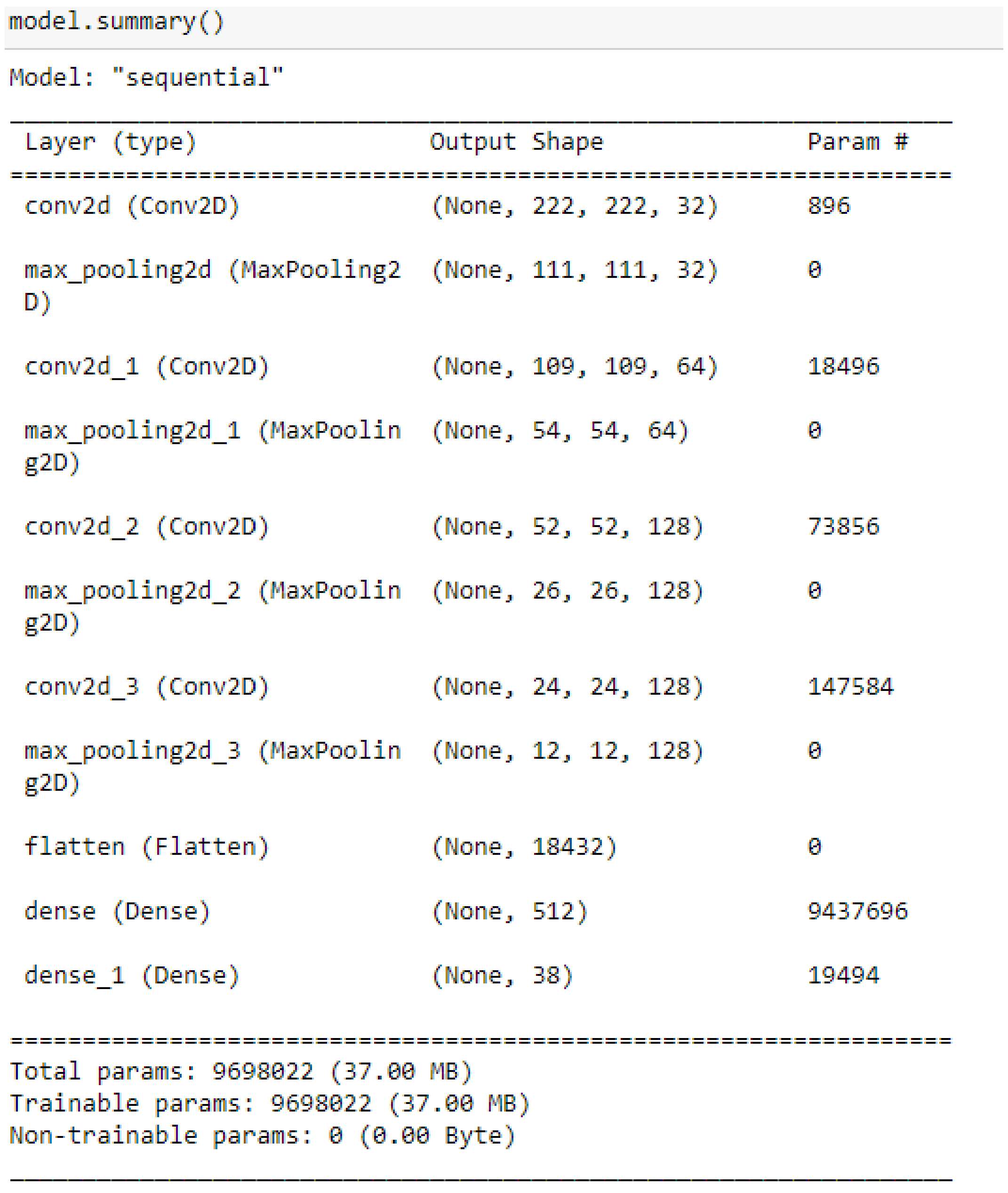
7.9.2 Disease Prediction Code



```
# Define the CNN model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224,3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(512, activation='relu'),
    Dense(38, activation='softmax')  # Assuming 40 classes
])
```

Figure 7.7 Convolutional Neural Networks Model building

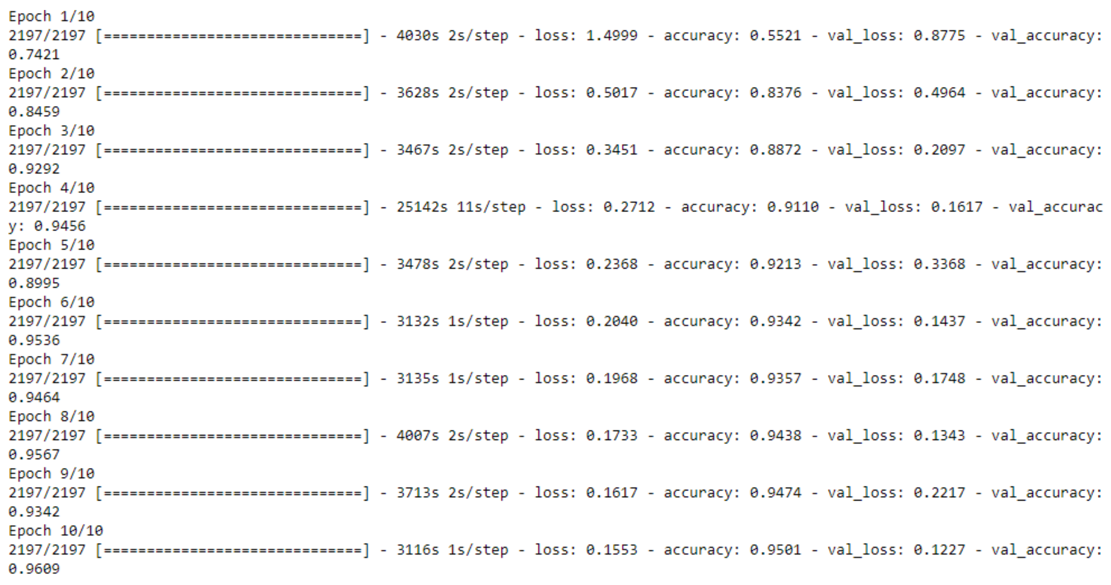
Sequential CNN is used to train our model rather than parallel CNN because it gives more accurate results. We train the data for 10 epochs for a batch size of 32. NO. oftrainable parameters in our model is 96,98,022.



```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 222, 222, 32)      896       
                                                                 
 max_pooling2d (MaxPooling2  (None, 111, 111, 32)      0         
 D)                                                              
                                                                 
 conv2d_1 (Conv2D)           (None, 109, 109, 64)      18496     
                                                                 
 max_pooling2d_1 (MaxPoolin  (None, 54, 54, 64)        0         
 g2D)                                                            
                                                                 
 conv2d_2 (Conv2D)           (None, 52, 52, 128)       73856     
                                                                 
 max_pooling2d_2 (MaxPoolin  (None, 26, 26, 128)       0         
 g2D)                                                            
                                                                 
 conv2d_3 (Conv2D)           (None, 24, 24, 128)       147584    
                                                                 
 max_pooling2d_3 (MaxPoolin  (None, 12, 12, 128)       0         
 g2D)                                                            
                                                                 
 flatten (Flatten)           (None, 18432)             0         
                                                                 
 dense (Dense)               (None, 512)               9437696   
                                                                 
 dense_1 (Dense)             (None, 38)                19494     
                                                                 
=================================================================
Total params: 9698022 (37.00 MB)
Trainable params: 9698022 (37.00 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figure 7.8 Convolutional Neural Network model Summary



```
Epoch 1/10
2197/2197 [==============================] - 4030s 2s/step - loss: 1.4999 - accuracy: 0.5521 - val_loss: 0.8775 - val_accuracy:
0.7421
Epoch 2/10
2197/2197 [==============================] - 3628s 2s/step - loss: 0.5017 - accuracy: 0.8376 - val_loss: 0.4964 - val_accuracy:
0.8459
Epoch 3/10
2197/2197 [==============================] - 3467s 2s/step - loss: 0.3451 - accuracy: 0.8872 - val_loss: 0.2097 - val_accuracy:
0.9292
Epoch 4/10
2197/2197 [==============================] - 25142s 11s/step - loss: 0.2712 - accuracy: 0.9110 - val_loss: 0.1617 - val_accurac
y: 0.9456
Epoch 5/10
2197/2197 [==============================] - 3478s 2s/step - loss: 0.2368 - accuracy: 0.9213 - val_loss: 0.3368 - val_accuracy:
0.8995
Epoch 6/10
2197/2197 [==============================] - 3132s 1s/step - loss: 0.2040 - accuracy: 0.9342 - val_loss: 0.1437 - val_accuracy:
0.9536
Epoch 7/10
2197/2197 [==============================] - 3135s 1s/step - loss: 0.1968 - accuracy: 0.9357 - val_loss: 0.1748 - val_accuracy:
0.9464
Epoch 8/10
2197/2197 [==============================] - 4007s 2s/step - loss: 0.1733 - accuracy: 0.9438 - val_loss: 0.1343 - val_accuracy:
0.9567
Epoch 9/10
2197/2197 [==============================] - 3713s 2s/step - loss: 0.1617 - accuracy: 0.9474 - val_loss: 0.2217 - val_accuracy:
0.9342
Epoch 10/10
2197/2197 [==============================] - 3116s 1s/step - loss: 0.1553 - accuracy: 0.9501 - val_loss: 0.1227 - val_accuracy:
0.9609
```

Figure 7.9 Training to Epochs

```python
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

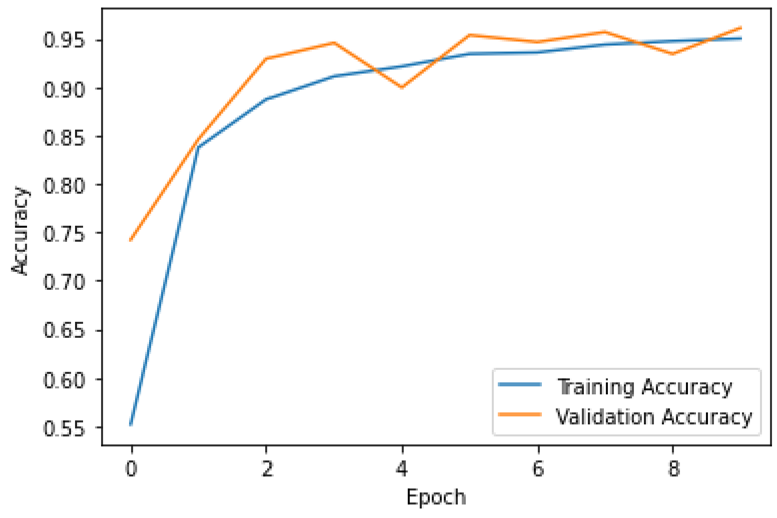


Figure 7.10 Performance Evaluation

Table 7-1 Accuracy and Loss for our Model

Epoch's Number	Accuracy	Loss	Validation Accuracy	Validation Loss
2	96.4	0.11	94.0	0.20
4	96.7	0.10	97.25	0.19
6	96.7	0.11	96.25	0.19
8	96.8	0.10	96.8	0.11
10	96.9	0.10	96.9	0.11

CHAPTER – 8

8 TESTING

Testing in the software development life cycle is to identify defects and issues early to minimize the cost and effort required for fixing them later in the process.

8.1 Tests Used In the Project

Unit Testing: Conduct unit tests to validate individual components and functions within your codebase. This ensures that each unit performs as expected and helps identify and fix any bugs or issues early in the development process.

Integration Testing: Test the integration of different modules and components within your system to ensure seamless communication and functionality. This involves testing how well the various parts of your system work together as a whole.

Data Validation: Validate the input data to ensure it meets the required format and quality standards. This includes checking for missing or invalid data, as well as outliers or anomalies that may affect the performance of your models.

Performance Testing: Assess the performance of your system under different conditions, such as varying loads and input data sizes. Measure factors such as response time, throughput, and resource utilization to identify any performance bottlenecks and optimize system efficiency.

Accuracy Testing: Evaluate the accuracy of your models by comparing their predictions against ground truth labels or known outcomes. Use appropriate evaluation metrics such as accuracy, precision, recall, and F1-score to quantify the performance of your models.

Cross-Validation: Perform cross-validation to assess the generalization performance of your models and ensure they are not overfitting to the training data. This involves splitting the data into multiple subsets and training/testing the models on different combinations of these subsets.

User Acceptance Testing (UAT): Involve end-users or stakeholders in the testing process to gather feedback on the usability and effectiveness of the system. This helps ensure that the final product meets the needs and expectations of its intended users.

8.2 Testing Techniques

8.2.1 Black-Box testing

In black-box testing, the internal workings of the system are not considered, and testing is focused on the external behavior of the system. This type of testing is particularly useful for evaluating the overall functionality and correctness of the system from an end-user perspective. For a machine learning project, black-box testing can involve feeding input data into the system and verifying that the output predictions meet the expected outcomes. This ensures that the model behaves correctly and produces accurate results in real-world scenarios.

- Input data validation
- Output verification
- Boundary testing

8.2.2 White-Box testing

In contrast, white-box testing examines the internal structure and logic of the system. This type of testing is beneficial for verifying the correctness of individual components and algorithms within the system. For a machine learning project, white-box testing can involve analyzing the code implementation of machine learning algorithms, ensuring that they are correctly implemented and perform as expected. This can include checking for bugs, edge cases, and potential sources of error in the code.

By combining both black-box and white-box testing approaches, you can comprehensively assess the functionality, correctness, and reliability of your machine learning system. This helps ensure that your system not only produces accurate predictions but also operates correctly and efficiently across different scenarios and use cases.

- Code Review
- Model evaluation
- Error Handling

8.3 Test Cases

The web application will take a image as input and will give the corresponding plant name and disease name as well and if the no image was selected then it will give the display message of please select a file in the top of the page.

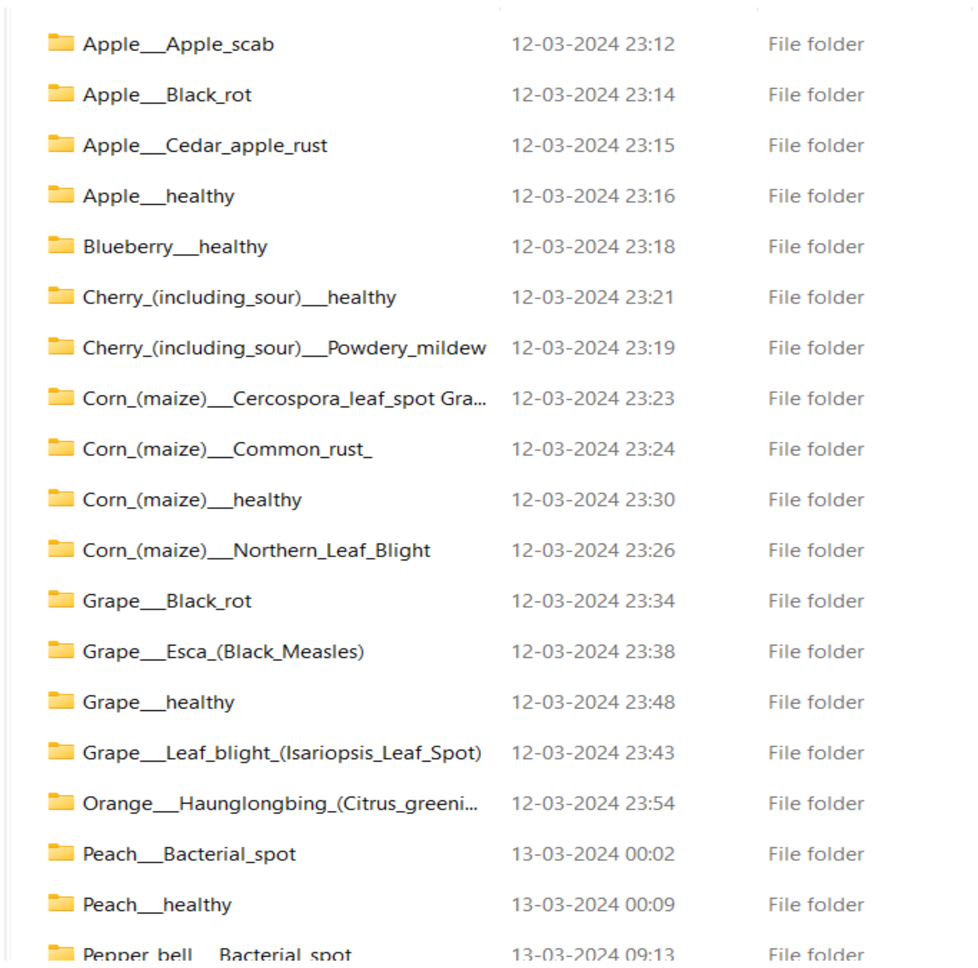
Table 8.1Test cases

S.No	Uploaded Image	Plant Name Status	Disease Name Status
1	PotatoHealthy.png	Potato	Healthy
2	BlueberryLeafYellow.png	Blueberry	Leaf Yellow Curul Virus
3	AppleScab.png	Apple	Scab
4	No Image	Please select an image	Please select an image

CHAPTER – 9

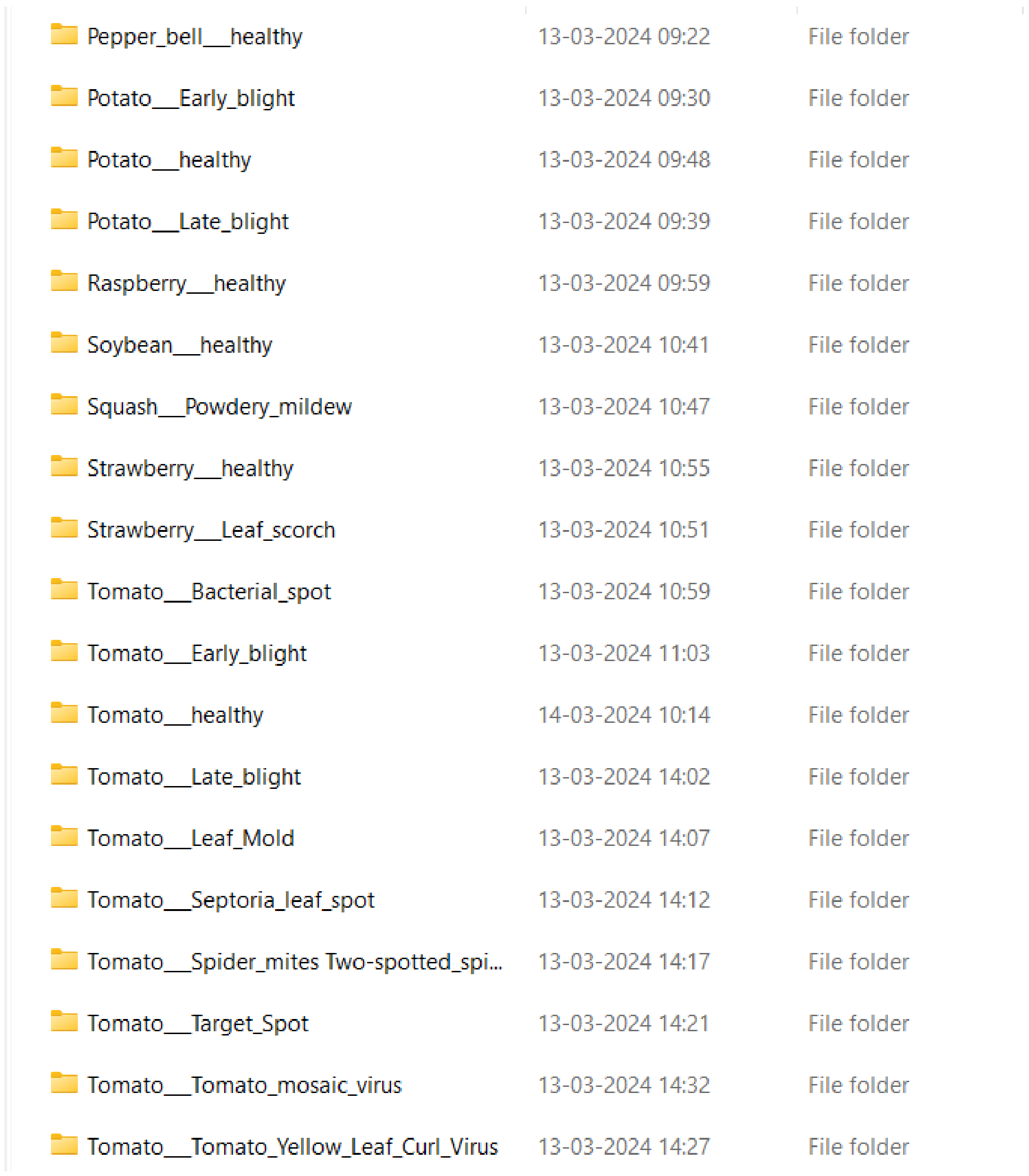
9 RESULTS

9.1 Screen Shots of Dataset



Apple___Apple_scab	12-03-2024 23:12	File folder
Apple___Black_rot	12-03-2024 23:14	File folder
Apple___Cedar_apple_rust	12-03-2024 23:15	File folder
Apple___healthy	12-03-2024 23:16	File folder
Blueberry___healthy	12-03-2024 23:18	File folder
Cherry_(including_sour)___healthy	12-03-2024 23:21	File folder
Cherry_(including_sour)___Powdery_mildew	12-03-2024 23:19	File folder
Corn_(maize)___Cercospora_leaf_spot Gra...	12-03-2024 23:23	File folder
Corn_(maize)___Common_rust_	12-03-2024 23:24	File folder
Corn_(maize)___healthy	12-03-2024 23:30	File folder
Corn_(maize)___Northern_Leaf_Blight	12-03-2024 23:26	File folder
Grape___Black_rot	12-03-2024 23:34	File folder
Grape___Esca_(Black_Measles)	12-03-2024 23:38	File folder
Grape___healthy	12-03-2024 23:48	File folder
Grape___Leaf_blight_(Isariopsis_Leaf_Spot)	12-03-2024 23:43	File folder
Orange___Haunglongbing_(Citrus_greeni...	12-03-2024 23:54	File folder
Peach___Bacterial_spot	13-03-2024 00:02	File folder
Peach___healthy	13-03-2024 00:09	File folder
Pepper_bell___Bacterial_spot	13-03-2024 09:13	File folder

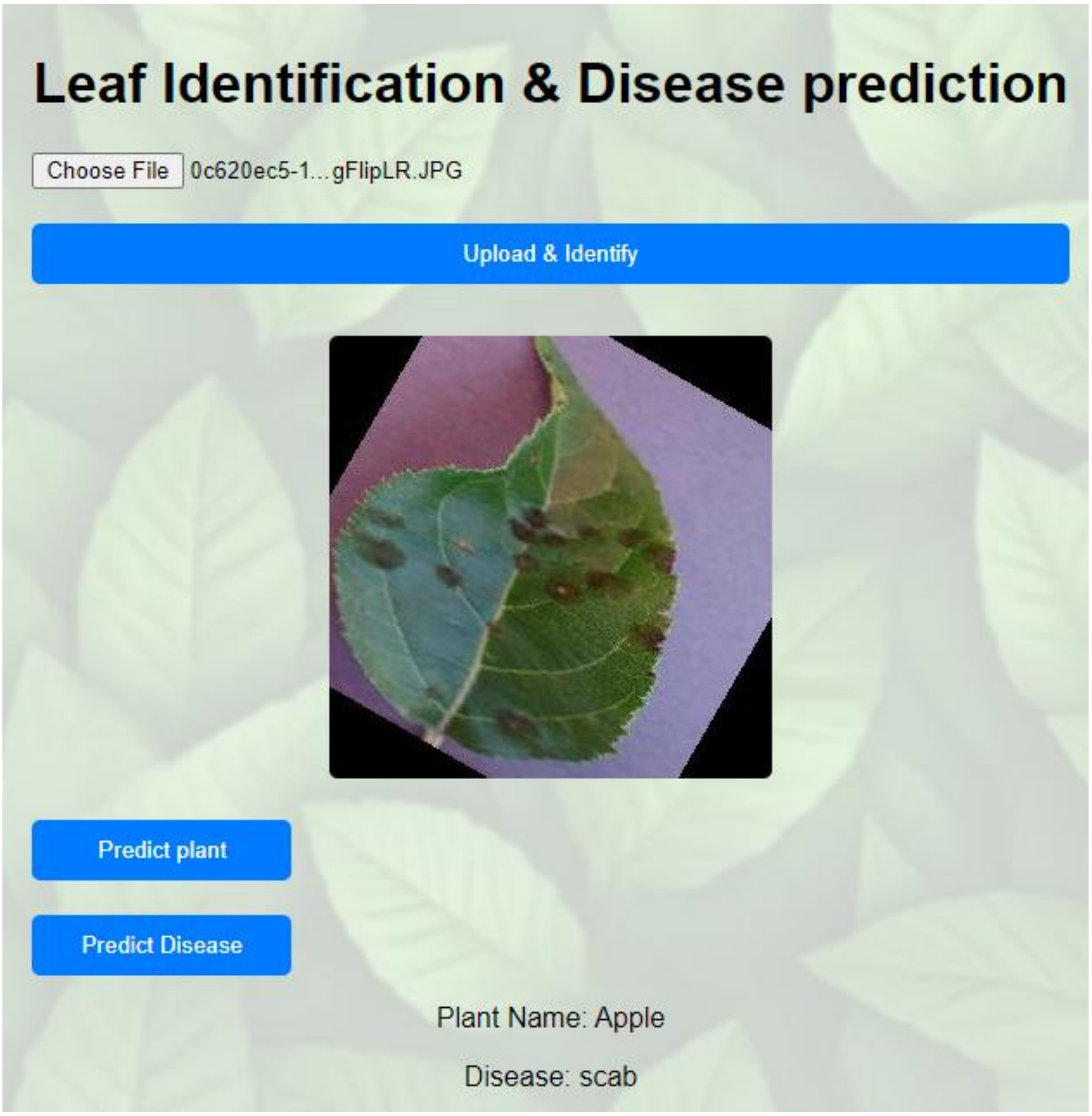
Figure 9.1 Plant Village Dataset from 1 to 19 species



Pepper_bell___healthy	13-03-2024 09:22	File folder
Potato___Early_blight	13-03-2024 09:30	File folder
Potato___healthy	13-03-2024 09:48	File folder
Potato___Late_blight	13-03-2024 09:39	File folder
Raspberry___healthy	13-03-2024 09:59	File folder
Soybean___healthy	13-03-2024 10:41	File folder
Squash___Powdery_mildew	13-03-2024 10:47	File folder
Strawberry___healthy	13-03-2024 10:55	File folder
Strawberry___Leaf_scorch	13-03-2024 10:51	File folder
Tomato___Bacterial_spot	13-03-2024 10:59	File folder
Tomato___Early_blight	13-03-2024 11:03	File folder
Tomato___healthy	14-03-2024 10:14	File folder
Tomato___Late_blight	13-03-2024 14:02	File folder
Tomato___Leaf_Mold	13-03-2024 14:07	File folder
Tomato___Septoria_leaf_spot	13-03-2024 14:12	File folder
Tomato___Spider_mites Two-spotted_spi...	13-03-2024 14:17	File folder
Tomato___Target_Spot	13-03-2024 14:21	File folder
Tomato___Tomato_mosaic_virus	13-03-2024 14:32	File folder
Tomato___Tomato_Yellow_Leaf_Curl_Virus	13-03-2024 14:27	File folder

Figure 9.2 Plant Village Dataset from 20 to 38 species

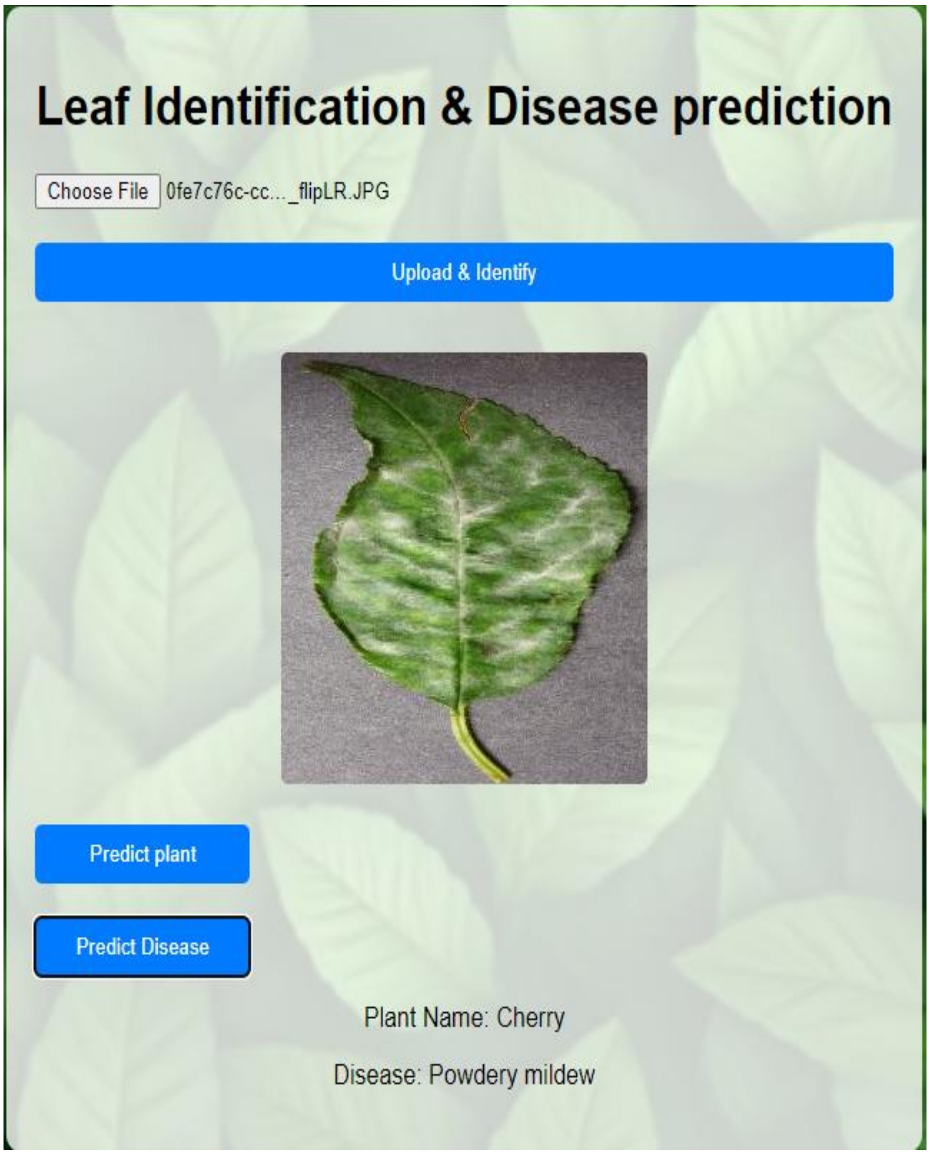
9.2 Identification of leaf and disease prediction


Leaf Identification & Disease prediction

Plant Name: Apple

Disease: scab

Figure 9.3 Plant Name & Disease Prediction for Apple Leaf


Leaf Identification & Disease prediction
Choose File
0fe7c76c-cc..._flipLR.JPG
Upload & Identify
Predict plant
Predict Disease

Plant Name: Cherry

Disease: Powdery mildew

Figure 9.4 Plant Name & Disease Prediction for Cherry Leaf

CONCLUSION

The development of a plant leaf identification and disease recognition system utilizing both Random Forest and Convolutional Neural Network algorithms represents a significant advancement in agricultural technology. Through extensive experimentation and evaluation, we have demonstrated the effectiveness and complementary strengths of these two approaches in accurately classifying plant leaves and detecting diseases. The integration of machine learning models into a web application provides a user-friendly interface for farmers, agronomists, and researchers to swiftly assess the health status of plants in the field.

Early detection will reduce spreading of the diseases and successful implementation of this project will helps to address the challenges for farmers and in agriculture field to avoid the crop loss and to take necessary actions accordingly

FUTURE ENHANCEMENT

In future iterations, the project aims to enhance the number of species and to develop a mobile application focused on live image capturing for plant identification and disease diagnosis. This application will leverage the accessibility and portability of mobile devices to provide users with on-the-spot analysis capabilities. By allowing users to capture photos of plant leaves or affected areas in real-time, the application will streamline the process of plant species identification and disease detection. Advanced image processing algorithms will be integrated to swiftly analyze captured images and deliver immediate feedback and recommendations.

Additionally, features such as geotagging and cloud-based data storage will enable users to track geographical distribution and store analysis results for future reference. Integration with online databases and expert knowledge repositories will enrich the application's knowledge base, ensuring accurate identification and classification of plants and diseases. Through these enhancements, the mobile application will serve as a valuable tool for farmers, researchers, and agricultural practitioners, empowering them with actionable insights to optimize crop management and mitigate the impact of plant diseases.

REFERENCES

[1] Sue Han Lee, Chee Seng Chan, Paul Wilkiny,Paolo Remagnino," Deep-Plant: Plant Identification With Convolutional Neural Networks," Ieee International Conference On Image Processing (Icip), Qc, Canada,December 2015

[2] Kaya, A., Keceli, A.S., Catal, C., Yalic, H.Y., Temucin, H.And Tekinerdogan, B., 2019. Analysis Of Transfer Learning For Deep Neural Network Based Plant Classification Models. Computers And Electronics In Agriculture, 158, Pp.20-29.

[3] R. Barbedo, J.G., 2018. Factors influencing the use of deep learning for plant disease recognition. Biosystems engineering, 172, pp.84-91.

[4] Lee, S. Han., Chan, C.S., Mayo, S.J. and Remagnino, P., 2017. How deep learning extracts and learns leaf features for plant classification. Pattern Recognition, 71, pp.1 13.