

PLANT DISEASE DETECTION USING MACHINE LEARNING

presented By

M.GURU MOUNIKA MCA II (23091F0009)

Under the Esteemed Guidance of

Mr. B. RAMESWARA REDDY

Asst . Professor, Dept of MCA.



DEPARTMENT OF MASTER OF COMPUTER APPLICATIONS
RAJEEV GANDHI MEMORIAL COLLEGE OF ENGINEERING&TECHNOLOGY
(AUTONOMOUS)
NANDYAL-518501(Estd-1995)

CONTENT

- ❖ Abstract
- ❖ Existing System & Disadvantages
- ❖ Proposed System & Advantages
- ❖ Literature review
- ❖ Data set
- ❖ Methodology
- ❖ UML representation
- ❖ Software and Hardware Requirements
- ❖ Modules
- ❖ Implementation & Algorithm
- ❖ Testing & Sample code
- ❖ Screenshots & Results
- ❖ Future Enhancements
- ❖ Conclusion

ABSTRACT

- ❖ Plant diseases have a significant impact on crop yields, causing economic losses for farmers and food shortages for communities. Early detection of diseases can help prevent their spread and minimize crop damage. Machine learning techniques offer a promising approach for automating disease detection in plants.
- ❖ In this study, we present a method for plant disease detection using machine learning algorithms. Our approach uses a dataset of images of diseased and healthy plants to train a Convolutional Neural Network (CNN) model. The trained model can then classify new images as either diseased or healthy.
- ❖ Our method provides a practical and effective approach for plant disease detection, which can be used to monitor crops and identify potential diseases early on.

EXISTING SYSTEM

- 1. Traditional approach :** This model supported standard metrics and gave decent accuracy score on the tested datasets but it held few drawbacks like involving more computational resources and high processing time, as the whole model had to be trained on all data points, absence of transfer learning is main drawback.
- 2. Inception V3-based approach :** Inception V3 is a transfer learning technique that is a 42-layer deep learning network with fewer parameters. We used the Inception V3 pretrained model to perform transfer learning on our 10-class tomato disease problem. We found the accuracy is just above than VGG16 and comes to 85% when we trained for 10 epochs on google colab. Resulting in faster pre-processing, low computational resources, processing time, and training time, resulting in an efficient model.

DISADVANTAGES

❖ Data Quality:

The effectiveness of machine learning models heavily relies on the and quantity of data. Poor data can lead to inaccurate predictions.

❖ Cost:

Initial setup costs for technology and data collection can be high, especially for small-scale farmers.

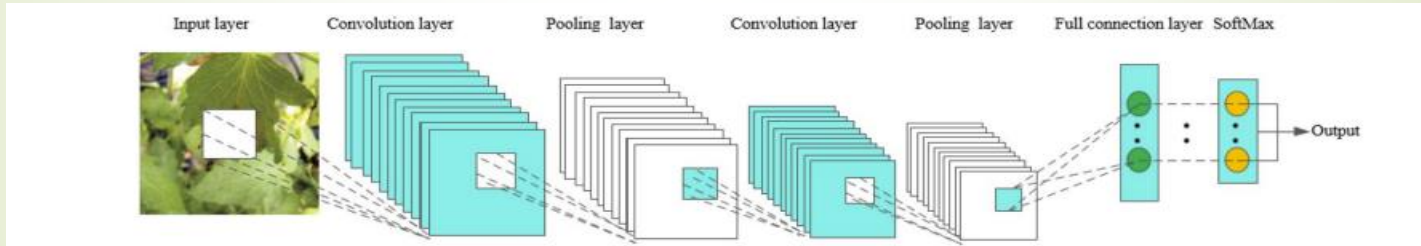
❖ Dependency on Technology:

Farmers may become reliant on technology and data, potentially neglecting traditional methods and knowledge.

PROPOSED SYSTEM

CNN(Convolutional Neural Network) using MobileNetV2:

- A Convolutional Neural Network (CNN) is a type of deep learning neural network that is commonly used for image recognition, classification, and other computer vision tasks.



- In our work, we have used model known as **MobileNetV2** . It is a 53 layers deep CNN(Convolutional Neural Network) and takes input images of size 128x128 pixels.

ADVANTAGES

❖ **Accuracy and Speed:**

Machine learning algorithms can analyze large datasets quickly, often providing more accurate disease detection than traditional methods.

❖ **Scalability:**

ML models can easily scale to analyze data from many plants, making them suitable for large agricultural operations.

❖ **Early Detection:**

ML can identify subtle signs of disease early, allowing for timely intervention and potentially reducing crop loss.

LITERATURE REVIEW

Title	Authors	Publication Year	Algorithm Used	Disadvantages
Segmentation of Plant Disease Spots Using Automatic SRG Algorithm: A Look Up Table Approach	Rajat Kanti Sarkar, Ankita Pramanik	2022	Automatic SRG	Difficulty with Complex Plant Structures, Can't deal with large datasets
A Machine Learning Technique for Identification of Plant Diseases in Leaves	Deepa, Rashmi N, Chinmai Shetty	2023	SVM	Less Accuracy
Disease Detection Using Transfer Learning In Coffee Plants	Lakshay Datta, Ashwani Kumar Rana	2021	CNN using VGG 16	Less Accuracy in training large datasets

DATA SET

- ❖ The data set contains leaves that are both diseased and diseased free we have collected the data set from Kaggle.
- ❖ The total no.of images is 11000, which have been affected by different biotech stresses. This dataset was split into two parts that are the training dataset and val set.
- ❖ The training set contains 1000 images which is 75% of the data taken.
- ❖ The validation set is the data set that is used to evaluate the performance of the trained model. The validation set contains 1000 images which is 25% of the dataset.

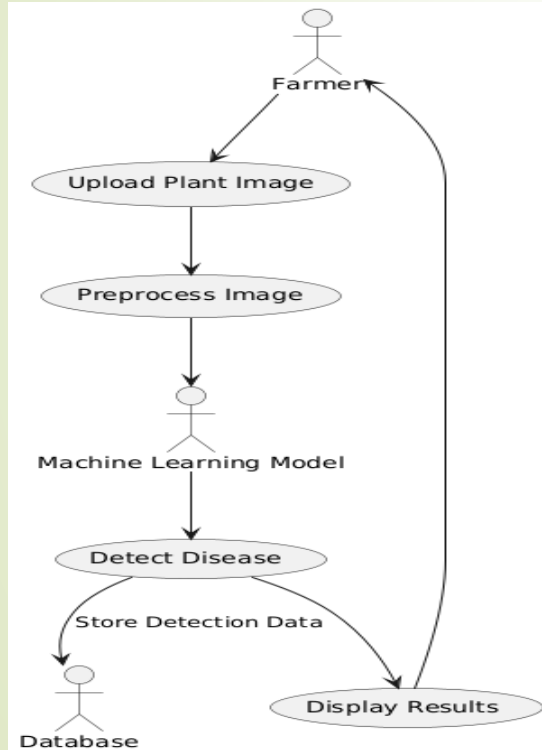
Image_ID	Plant_Type	Disease_Name	Image_Path/URL	Symptom_Description	Severity_Level (0-5)	Date_Captured	Location
1	Tomato	Late Blight	/path/to/image1.jpg	Dark lesions on leaves and stems	4	2023-06-15	California, USA
2	Potato	Early Blight	/path/to/image2.jpg	Circular brown spots on leaves	3	2023-07-22	Punjab, India
3	Apple	Apple Scab	/path/to/image3.jpg	Dark, scabby spots on leaves & fruit	2	2023-05-10	Ontario, Canada
4	Wheat	Powdery Mildew	/path/to/image4.jpg	White, powdery growth on leaves	3	2023-04-01	Kansas, USA
5	Tomato	Septoria Leaf Spot	/path/to/image5.jpg	Circular black spots with yellow halos	4	2023-06-30	Queensland, Australia
6	Grape	Black Rot	/path/to/image6.jpg	Dark, shriveled spots on leaves	5	2023-08-11	Burgundy, France
7	Citrus	Citrus Canker	/path/to/image7.jpg	Brown lesions with yellow halos	4	2023-07-18	Florida, USA
8	Corn	Northern Leaf Blight	/path/to/image8.jpg	Long, gray-green lesions on leaves	3	2023-09-03	São Paulo, Brazil

METHODOLOGY

- ❖ **Data collection:** Collect a large dataset of images of healthy and diseased plants. The dataset is representative of the different types of plants and diseases that you want to detect. Here we are considering New Plant Village Dataset from Kaggle.
- ❖ Import the necessary libraries .
- ❖ Mount Google drive to Google colab Notebook.
- ❖ **Data preprocessing and Augmentation:** Preprocess the images to normalize the pixel values, resize the images to a uniform size. Augment the dataset by applying random transformations to the images, such as rotation of images.
- ❖ Train the model using training dataset contain of 10000 images.
- ❖ Test the model using diverse test examples in validation dataset contain of 1000 images.
- ❖ Observe the accuracy of the model.

UML REPRESENTATION

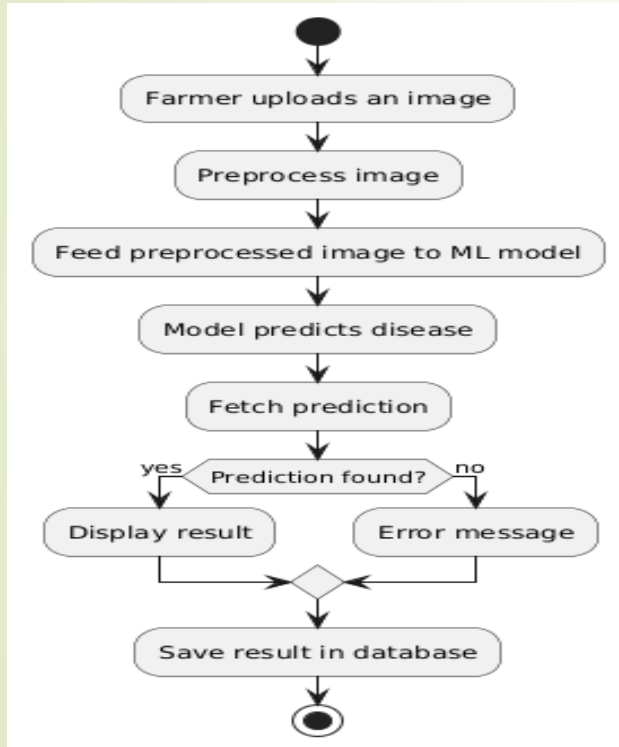
Use case diagram



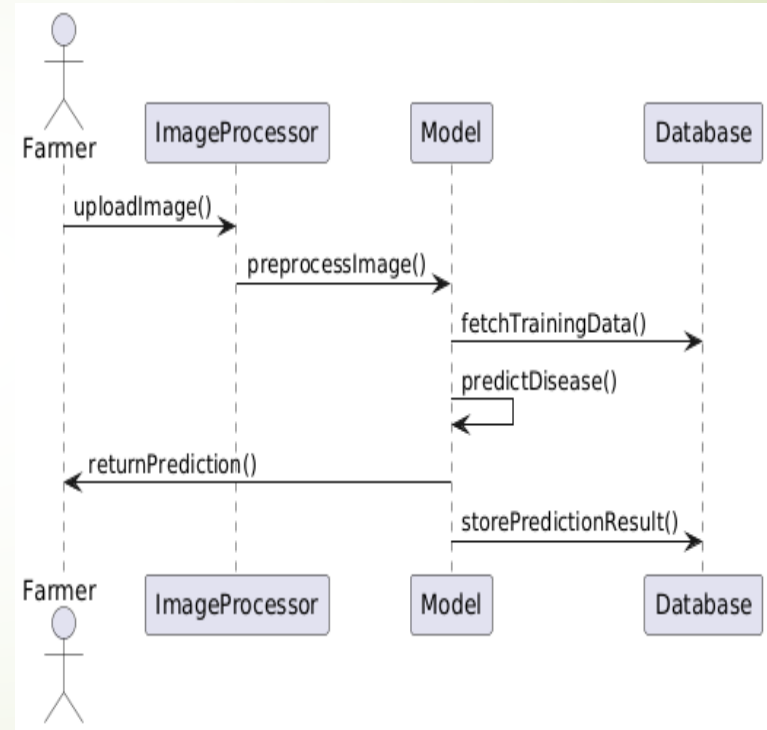
Class diagram



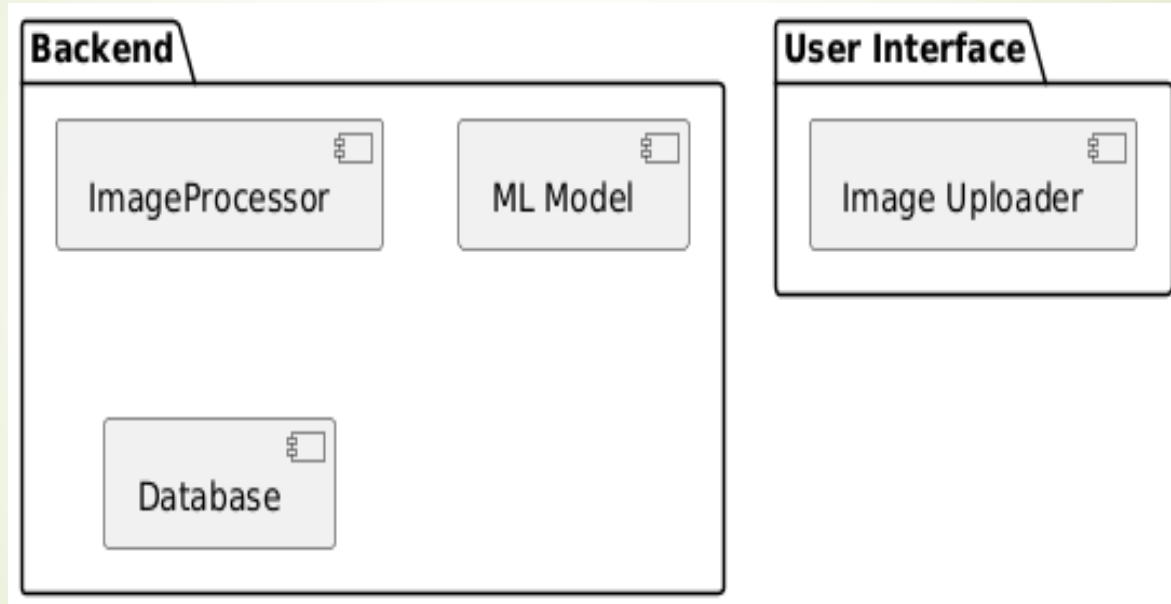
Activity diagram



Sequence diagram



Component diagram



SOFTWARE REQUIREMENTS

- ❖ Anaconda Navigator
- ❖ Python
- ❖ Jupyter Notebook
- ❖ Required Libraries
- ❖ Google Colab

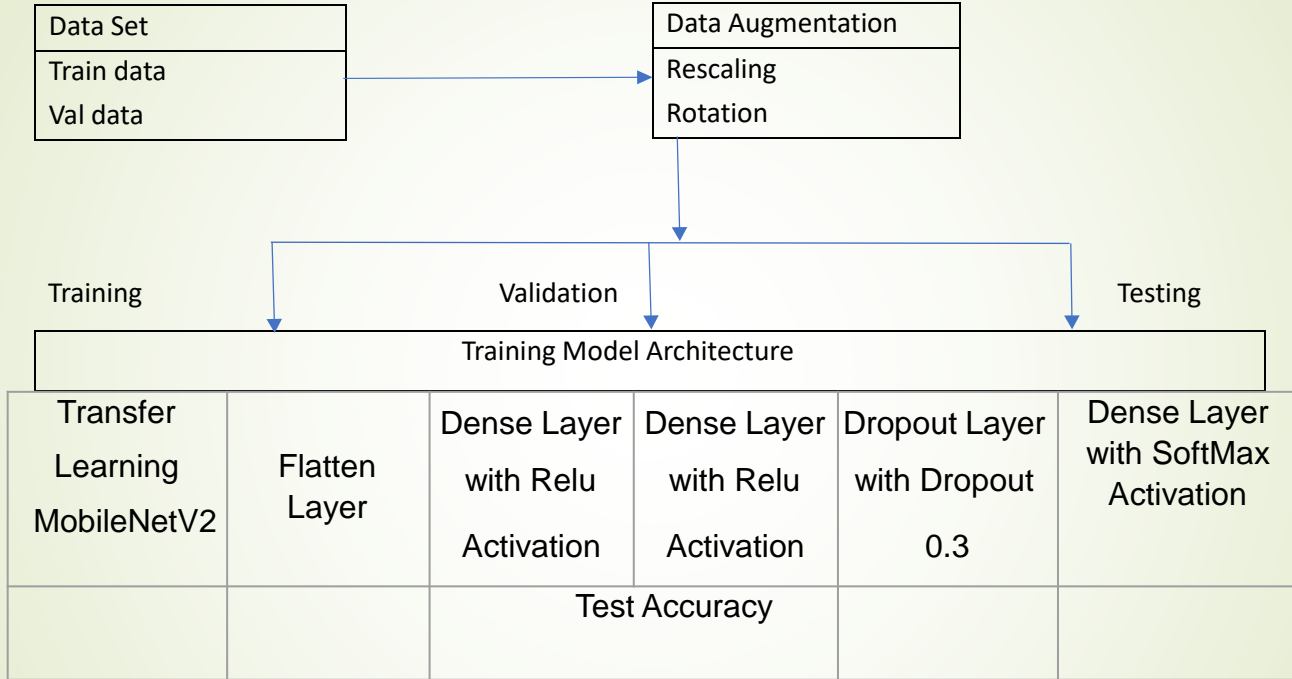
HARDWARE REQUIREMENTS

- ❖ Processor : Intel Core i5/i7
- ❖ Storage : 512 GB
- ❖ RAM : 8+ GB

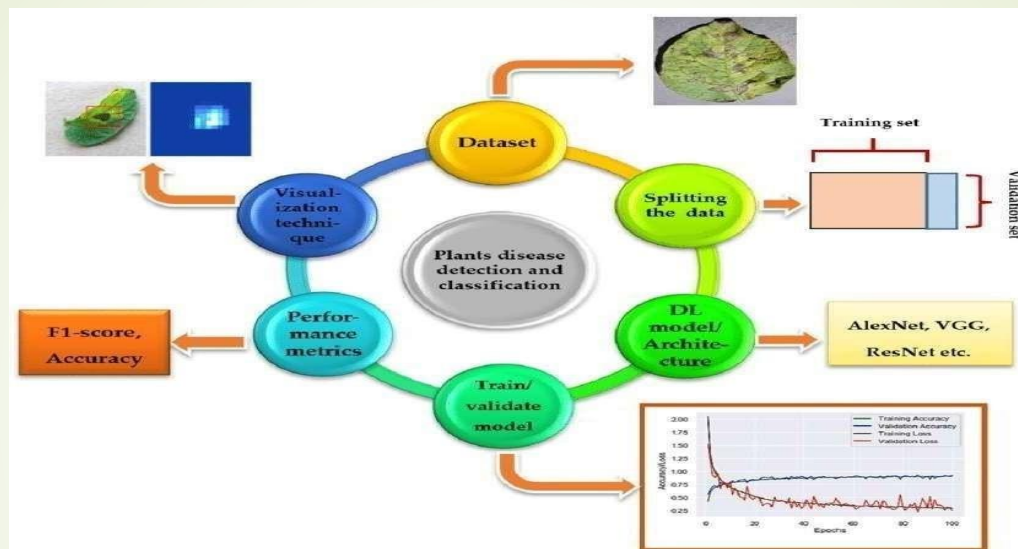
ALGORITHM

- MobileNetV2 Architecture is considered our base model because it outperforms other architectures like inception v3 and VGG 16 and other architectures over the images.
- MobileNetV2 is a transfer learning architecture designed for mobile and embedded devices with limited computational resources. It is a lightweight convolutional neural network (CNN) that achieves a good balance between accuracy and efficiency.
- MobileNetV2 consists of depth wise separable convolutions, which decompose the standard convolution operation into two separate layers: depth wise convolution and pointwise convolution. The depth wise convolution applies a single filter to each input channel independently, reducing the computational cost. The pointwise convolution then performs a 1×1 convolution to combine the output channels from the depth wise convolution.
- MobileNetV2 introduces a new component called "inverted residuals" that further improves efficiency.
- Inverted residuals utilize bottleneck blocks with a bottleneck layer that reduces the number of input channels before applying depth wise separable convolutions.
- This reduces the computational burden while preserving important features. To adapt MobileNetV2 for transfer learning, the architecture can be used as a feature extractor.

SYSTEM ARCHITECTURE



MobileNetV2 Model Architecture



CNN Architecture

MODULES

➤ Image Processing:

It involves various tasks such as resizing, normalization, noise removal, and enhancing image quality to prepare them for further analysis. Techniques like edge detection, image segmentation, and color space conversion may also be applied to extract relevant information from the images. This module ensures that the input images are appropriately pre-processed to improve the accuracy of disease detection algorithms.

➤ Feature Extraction:

Feature extraction module focuses on identifying extracting meaningful features from pre-processed images. It involves analyzing the texture, shape, and other visual characteristics of the images to represent them in a more compact and informative manner. The objective of this module is to convert unprocessed image data into a feature vector suitable for utilization in machine learning algorithms.

➤ Classification:

The classification module is responsible for categorizing the pre-processed and feature extracted images into different disease classes. It utilizes machine learning algorithms such as support vector machines (SVM), k-nearest neighbors (k-NN), or deep learning models like CNNs for accurate classification. The module trains the classifier using labeled data from the dataset and then predicts the disease class of unseen images based on their extracted features.

➤ User Interface:

The user interface module provides an interactive platform for users to interact with the system. It includes functionalities for uploading input images, selecting disease detection options, and displaying the results of classification. Graphical components like buttons, text boxes, and image displays are utilized to enrich user interaction and simplify navigation. The System Requirements Specification (SRS), also referred to as Software Requirements Specification, comprises a document or collection of documents outlining the functionalities and operations of a system or software application.

IMPLEMENTATION

➤ Python:

Python, recognized for its simplicity and readability, is a versatile programming language. It is interpreted, object-oriented, and high-level, making it suitable for various applications. Python's design philosophy emphasizes code readability, employing whitespace indentation for code blocks. This design choice, along with its concise syntax, enables programmers to express concepts with fewer lines of code compared to languages like C++ or Java. Python provides support for various programming paradigms, encompassing object-oriented, imperative, Python supports multiple programming paradigms, including object-oriented, imperative, functional, and procedural, providing flexibility for developers.

➤ Machine Learning Libraries:

Scikit-Learn:

Scikit-learn stands as a robust Python library for machine learning that supports a wide variety of machine learning algorithms. It is known for its simple and efficient tools for data analysis and data mining. Perfect for beginners and advanced users, scikit-learn is ideal for implementing machine learning algorithms quickly and effectively.

➤ Keras:

Keras, an open-source software library, offers a Python interface for artificial neural networks, serving as a bridge for the TensorFlow library. Up until version 2.3, Keras also supported other backends such as Theano and Microsoft Cognitive Toolkit (CNTK). Keras simplifies many tasks in creating deep learning models and is designed to facilitate fast experimentation with deep neural networks. It is focused on being user- friendly, modular, and extensible.

➤ Tensor Flow:

Tensor Flow is a complete open-source platform for machine learning created by Google. It offers a broad and adaptable ecosystem of tools, libraries, and community resources, enabling researchers to advance the state-of-the-art in ML and developers to easily create and deploy models ML- powered applications. Tensor Flow is designed to facilitate the development of large-scale neural networks with many layers, primarily used for deep learning applications.

TESTING

➤ Testing

Testing aims to uncover errors by exercising software to ensure it meets requirements and user expectations. It involves checking the functionality of components, assemblies, and finished products to prevent unacceptable failures. Different test types address specific testing needs.

➤ Types Of Testing

■ Unit Testing:

Unit testing involves designing test cases to validate internal program logic and ensure that inputs produce valid outputs. It's done at the individual unit level after completion and before integration, focusing on specific business processes or system configurations. These tests validate each unique path of a process against documented specifications, ensuring accuracy and precise expected results.

■ Integration Testing:

Integration tests validate the functionality of integrated software components, ensuring they operate collectively as a single program. They focus on event-driven testing and the overall outcome of screens or fields. Integration tests verify that while individual components passed unit testing, their combination is correct and consistent, uncovering issues that may arise from component interactions.

■ **Functional Testing:**

Functional tests systematically demonstrate that functions tested meet specified business and technical requirements, system documentation, and user manuals. They focus on valid and invalid input classes, exercising identified functions, and exercising classes of application outputs. Functional testing also involves invoking interfacing systems or procedures. The organization and preparation of functional tests prioritize.

➤ **Test Strategy And Approach:**

Field testing will be executed manually, while functional tests will be meticulously scripted.

■ **Test objectives:**

- Ensure the functionality of all field entries.
- Validate the activation of pages from designated links.
- Verify timely response of entry screens and messages

SAMPLE CODE

```
▶ # Set seeds for reproducibility
import random
random.seed(0)

import numpy as np
np.random.seed(0)

import tensorflow as tf
tf.random.set_seed(0)
```

REQUIRED LIBRARIES

```
▶ import os
import json
from zipfile import ZipFile
from PIL import Image

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import layers, models
```

KAGGLE DATASET CREDENTIALS

```
▶ kaggle_credentials = json.load(open("kaggle.json"))
```

```
▶ # setup Kaggle API key as environment variables  
os.environ['KAGGLE_USERNAME'] = kaggle_credentials["username"]  
os.environ['KAGGLE_KEY'] = kaggle_credentials["key"]
```

```
▶ !pip install -q kaggle
```

UNZIPPING FILE

```
▶ # Unzip the downloaded dataset  
with ZipFile("plantvillage-dataset.zip", 'r') as zip_ref:  
    zip_ref.extractall()
```

```
▶ # Dataset Path  
base_dir = 'plantvillage dataset/color'
```

```
▶ print(os.listdir("plantvillage dataset"))  
  
print(len(os.listdir("plantvillage dataset/segmented")))  
print(os.listdir("plantvillage dataset/segmented")[:5])  
  
print(len(os.listdir("plantvillage dataset/color")))  
print(os.listdir("plantvillage dataset/color")[:5])  
  
print(len(os.listdir("plantvillage dataset/grayscale")))  
print(os.listdir("plantvillage dataset/grayscale")[:5])
```

TESTING FEATURES OF LEAF IMAGE

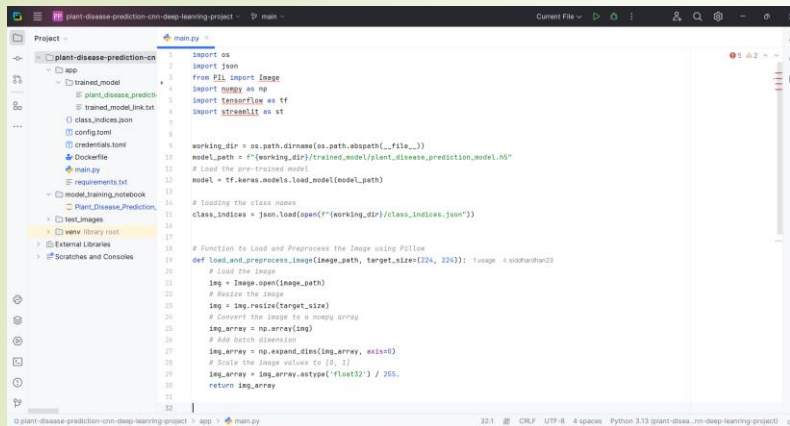
```
▶ image_path = """/content/plantvillage_dataset/color/Apple__Cedar_apple_rust  
/0321e067-d13b-47d0-b3a6-76ba6f357d02__FREC_C.Rust_3667.JPG""  
  
# Read the image  
img = mpimg.imread(image_path)  
  
print(img.shape)  
# Display the image  
plt.imshow(img)  
plt.axis('off') # Turn off axis numbers  
plt.show()
```

OUTPUT

(256, 256, 3)

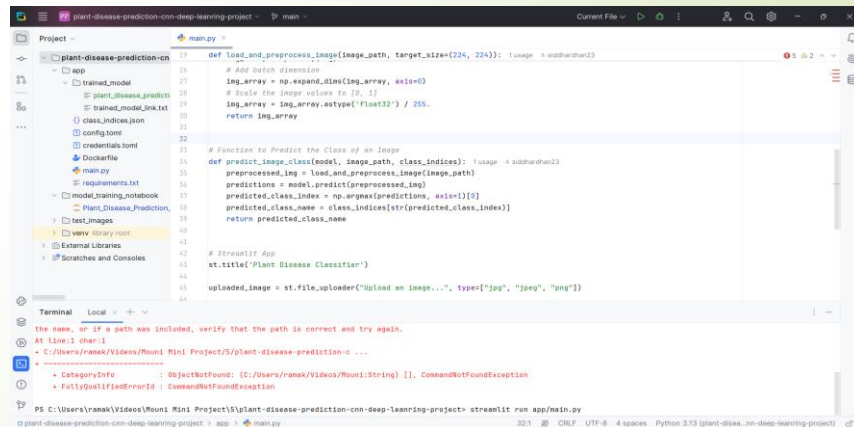


SCREENSHOTS & RESULTS



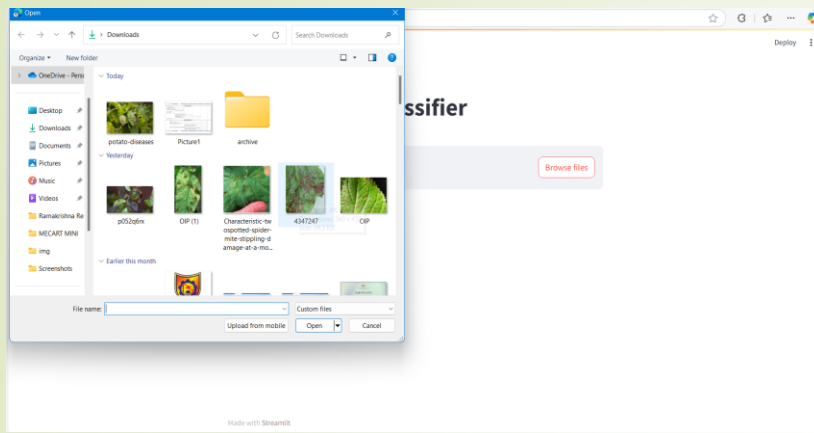
```
1 import os
2 import json
3 from PIL import Image
4 import numpy as np
5 import tensorflow as tf
6 import streamlit as st
7
8
9 working_dir = os.path.dirname(os.path.abspath(__file__))
10 model_path = f"{working_dir}/trained_model/plant_disease_prediction_model.h5"
11 # Load the pre-trained model
12 model = tf.keras.models.load_model(model_path)
13
14 # Loading the class names
15 class_indices = json.load(open(f"{working_dir}/class_indices.json"))
16
17
18 # Function to Load and Preprocess the Image using Pillow
19 def load_and_preprocess_image(image_path, target_size=(224, 224)):
20     # Load the image
21     img = Image.open(image_path)
22     # Resize the image
23     img = img.resize(target_size)
24     # Convert the image to a numpy array
25     img_array = np.array(img)
26     # Add batch dimension
27     img_array = np.expand_dims(img_array, axis=0)
28     # Scale the image values to [0, 1]
29     img_array = img_array.astype('float32') / 255.
30     return img_array
```

CODE IN PYCHARM

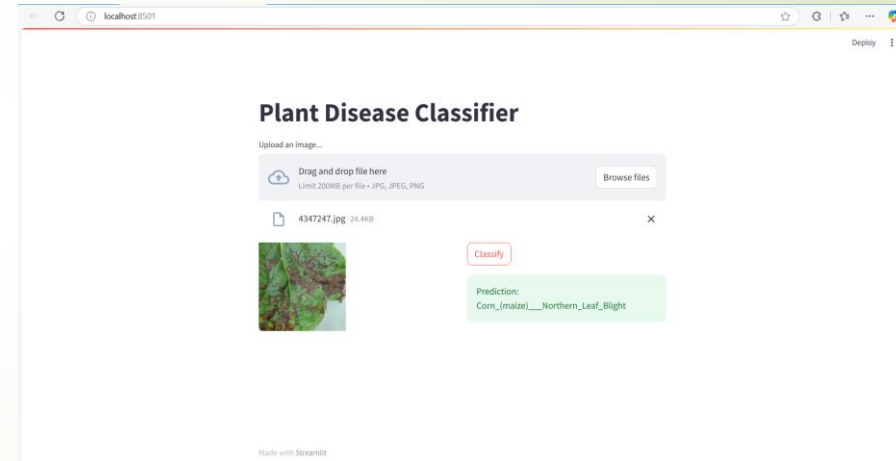


```
19 def load_and_preprocess_image(image_path, target_size=(224, 224)):
20     # Load the image
21     img = Image.open(image_path)
22     # Resize the image
23     img = img.resize(target_size)
24     # Convert the image to a numpy array
25     img_array = np.array(img)
26     # Add batch dimension
27     img_array = np.expand_dims(img_array, axis=0)
28     # Scale the image values to [0, 1]
29     img_array = img_array.astype('float32') / 255.
30     return img_array
31
32
33 # Function to Predict the Class of an Image
34 def predict_image(class_index, image_path, class_indices):
35     preprocessed_img = load_and_preprocess_image(image_path)
36     predictions = model.predict(preprocessed_img)
37     predicted_class_index = np.argmax(predictions, axis=-1)[0]
38     predicted_class_name = class_indices[str(predicted_class_index)]
39     return predicted_class_name
40
41
42 # Streamlit App
43 st.title('Plant Disease Classifier')
44
45 uploaded_image = st.file_uploader("Upload an image...", type=["jpg", "jpeg", "png"])
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

RUNNING CODE IN PYCHARM

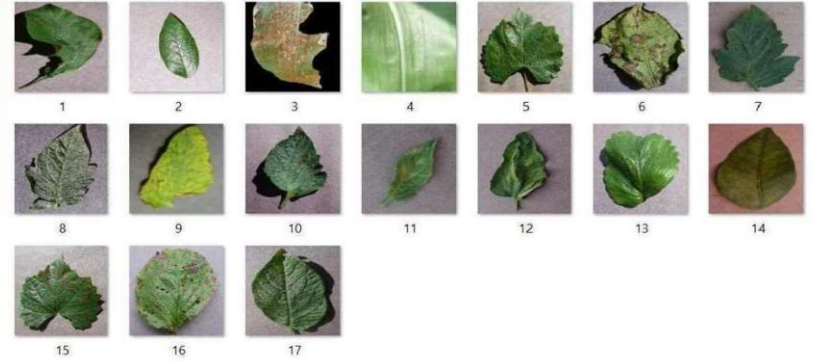
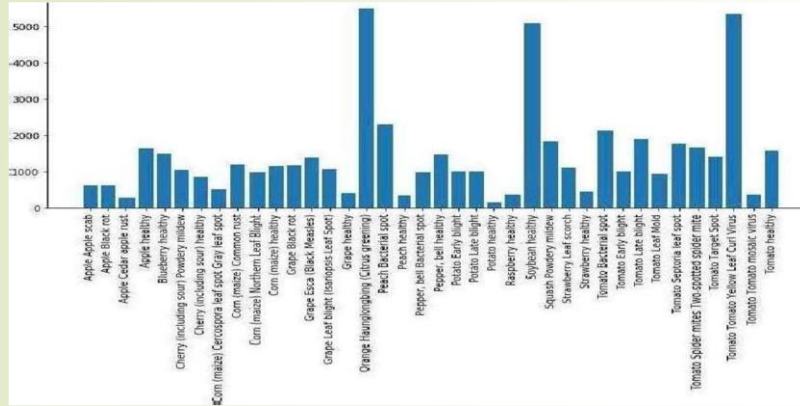


BROWSING PLANT LEAF IMAGES



PREDICTION OF THE PLANT LEAF

RESULTS



Sample input of data classes

Loading images for testing

FUTURE ENHANCEMENTS

- Future improvements could focus on improving deep learning models understanding of different types of images by training them with a wider variety of pictures from various sources. For classical machine learning, trying out different ways of picking out important information from images and testing new algorithms could help boost accuracy.
- Additionally, using new technologies designed specifically for speeding up calculations and making algorithms run more efficiently could make both deep learning and classical methods even better at analyzing images and spotting diseases in plants.

CONCLUSION

- Plant disease detection offers a transformative solution for addressing the challenges faced by agriculture, particularly in ensuring crop health and food security. By combining computer vision and machine learning methodologies, this approach enables early, non-destructive, and scalable detection of diseases, empowering farmers with timely information for effective management strategies.
- The advantages of Plant disease detection, including high throughput, cost- effectiveness, and data-driven insights, underscore its potential to revolutionize crop protection practices and support sustainable agricultural systems.
- As research and technology in this field continue to advance, image- based plant disease detection stands poised to play a pivotal role in enhancing crop productivity, resilience, and the sustainability of global food systems.
- Looking ahead, the continued advancement of image-based plant disease detection holds immense promise for transforming agricultural practices worldwide. As technology evolves and datasets grow, we can expect even greater accuracy and reliability in disease detection, empowering farmers to make data-driven decisions for crop protection.

THANK YOU!!!