PLANT DISEASE DETECTION

PROJECT REPORT

(Shivam Kabra – 20BAI10298
Divyansh Sahu – 20BAI10363
Gargi Choudhary – 20BAI10358
Rapeti Bhuvanananda Raviraj Jayanth – 20BAI10036)
(Ms. Kamya Paliwal)

TABLE OF CONTENTS

1.INTRODUCTION

1.1 Overview

The agricultural sector plays a vital role in ensuring food security and sustainability. However, plant diseases pose a significant threat to crop production. Early detection and accurate identification of plant diseases are essential for effective disease management strategies. The proposed project aims to develop a system that can detect and classify plant diseases based on images of plant leaves. By leveraging computer vision and machine learning algorithms, the system will provide farmers with a reliable tool for disease identification. The Leaf Disease Detection Flask App is a web application that provides an interface for users to upload images of leaves and receive a prediction of whether the leaves are healthy or diseased. The Flask App uses the deep learning model to make predictions on the uploaded images.

1.2 purpose

The aim of this project is to develop an intelligent system for plant disease detection. The detection of plant diseases at an early stage is crucial for effective disease management and preventing significant crop losses. Traditional methods of disease detection rely on visual observation by human experts, which can be time-consuming and prone to errors. This project proposes an automated approach using computer vision and machine learning techniques to accurately identify and classify plant diseases based on leaf images. The system will assist farmers in timely disease identification, enabling them to take appropriate measures to prevent the spread of diseases and protect their crops.

2.LITERATURE SURVEY

2.1 Existing Problem

One of the challenges in plant disease detection is providing a user-friendly and accessible platform for farmers to interact with the system. Traditional methods often involve complex setups or require technical expertise, making it difficult for farmers to utilize such systems effectively. Additionally, deploying and maintaining standalone applications can be cumbersome and limit accessibility.

2.2 Proposed Solution

To address these challenges, the proposed solution involves using the Flask framework to develop a web-based interface for plant disease detection. Flask is a lightweight and flexible web framework for Python, making it an ideal choice for creating user-friendly applications.

1. Problem: Limited accessibility and usability.

Solution: By developing a web application using Flask, farmers can access the plant disease detection system from any device with a web browser, including smartphones and tablets. This increases accessibility and usability, allowing farmers to easily upload images and receive disease diagnosis results.

2.Problem: Complex setup and deployment

Solution: Flask simplifies the setup and deployment process. It provides a development server that can be used for testing and debugging, and it supports various deployment options, such as hosting the application on a cloud platform or a local server. This makes it easier to deploy and maintain the plant disease detection system.

3.Problem: User interaction and feedback.

Solution: Flask enables interactive user interfaces, allowing farmers to interact with the system through forms, buttons, and other interactive components. It facilitates the display of disease diagnosis results in a user-friendly format, providing clear and concise information about the detected diseases and recommended actions.

4.Problem: Integration with other tools and services.

Solution: Flask can be easily integrated with other Python libraries, tools, and services. This enables seamless integration with image processing libraries, machine learning models, databases, and external APIs. For example, Flask can handle image uploads, pass the images to the disease detection algorithm, and retrieve the results for display to the user.

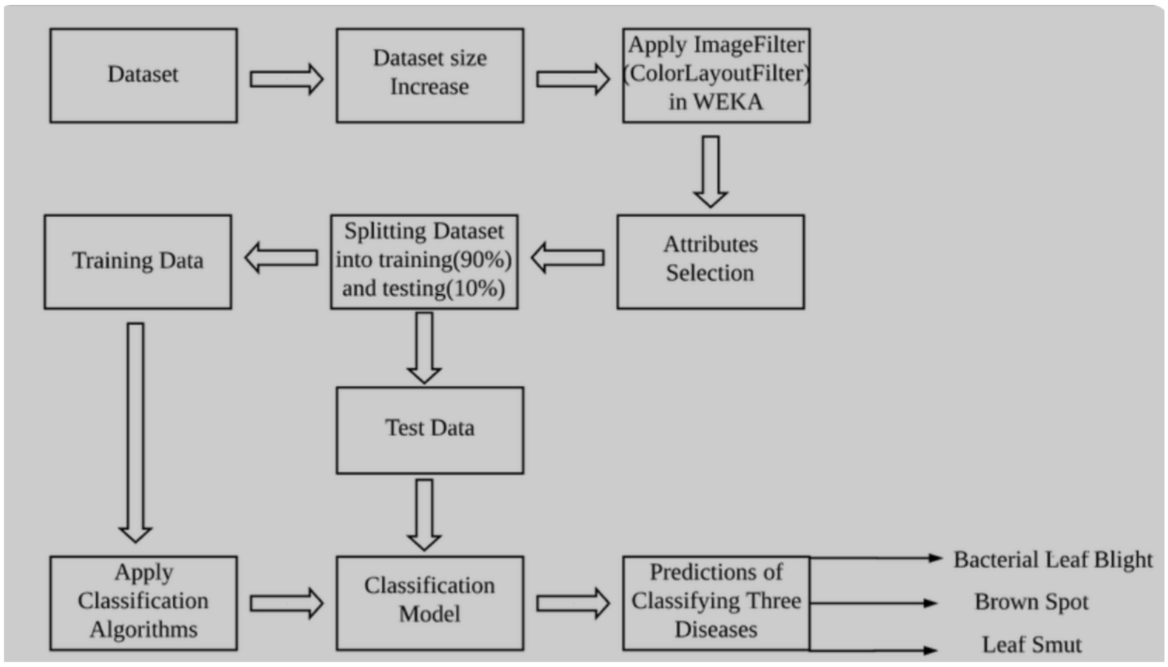
5.Problem: Scalability and future enhancements.

Solution: Flask supports modular and scalable application development. As the plant disease detection system evolves, additional features and functionalities can be easily incorporated into the Flask application, making it adaptable to future enhancements and expansions.

By utilizing Flask to develop a web-based interface, the proposed solution overcomes the limitations of traditional standalone applications. It provides a user-friendly, accessible, and scalable platform for farmers to detect and manage plant diseases effectively.

3.THEORITICAL ANALYSIS

3.1 Block Diagram



Dataset
Dataset size Increase
Apply ImageFilter (ColorLayoutFilter) in WEKA
Training Data
Splitting Dataset into training(90%) and testing(10%)
Attributes Selection
Test Data
Apply Classification Algorithms
Classification Model
Predictions of Classifying Three Diseases
Bacterial Leaf Blight
Brown Spot
Leaf Smut

3.2 Hardware / Software designing

To develop and deploy a plant disease detection system using Flask, you will need both hardware and software components. Here are the typical requirements for such a system:

Hardware Requirements:

1.Server or Hosting Platform: You will need a server or a hosting platform to run the Flask application. This can be a physical server, a virtual private server (VPS), or a cloud-based platform like AWS, Azure, or Google Cloud. The hardware specifications should be sufficient to handle the expected workload and traffic.

2.Storage: Sufficient storage capacity is required to store the Flask application code, the trained machine learning models, and the dataset if needed. Additionally, you may need storage space to handle user-uploaded images for disease detection.

3.Internet Connectivity: A stable and reliable internet connection is necessary for hosting the Flask application and allowing users to access the system from their devices.

Software Requirements:

1.Python: Flask is a Python web framework, so you will need to have Python installed on the server or hosting platform. Ensure that you have a compatible version of Python installed (e.g., Python 3.x).

2.Flask: Install the Flask framework using pip, the Python package manager. This can be done by running the command pip install flask in the terminal.

3.Required Python Libraries: Depending on your specific implementation, you may need additional Python libraries for tasks such as image processing, machine learning, and database management. Some commonly used libraries include NumPy, OpenCV, TensorFlow, Keras, and SQLAlchemy. Install these libraries using pip as needed.

4.Machine Learning Framework: If you plan to use machine learning algorithms for disease detection, you may need a machine learning framework like TensorFlow or PyTorch. Install the necessary framework and its dependencies according to the documentation.

5.Image Processing Libraries: Image processing tasks, such as preprocessing the uploaded leaf images, may require libraries like Pillow or OpenCV. Install the required libraries using pip.

6.Web Server: A web server is needed to host the Flask application. Popular web servers for Flask include Gunicorn and uWSGI. You can choose and configure the web server according to your hosting environment and preferences.

4.EXPERIMENTAL INVESTIGATIONS

During the development of a plant disease detection solution using Flask, several analyses and investigations are typically conducted. Here are some key areas that are commonly explored:

1.Dataset Analysis:Analyzing the dataset is crucial to understand its composition, distribution, and quality. Investigate the number of samples for each class (healthy and diseased), potential class imbalances, and the diversity of plant species and diseases covered. This analysis helps ensure that the dataset is representative and suitable for training the disease detection model.

2.Preprocessing Techniques:Investigation of various preprocessing techniques is necessary to enhance the quality and consistency of the leaf images. Experiment with resizing, normalization, noise removal, and color correction methods to determine which techniques work best for improving the accuracy of disease detection.

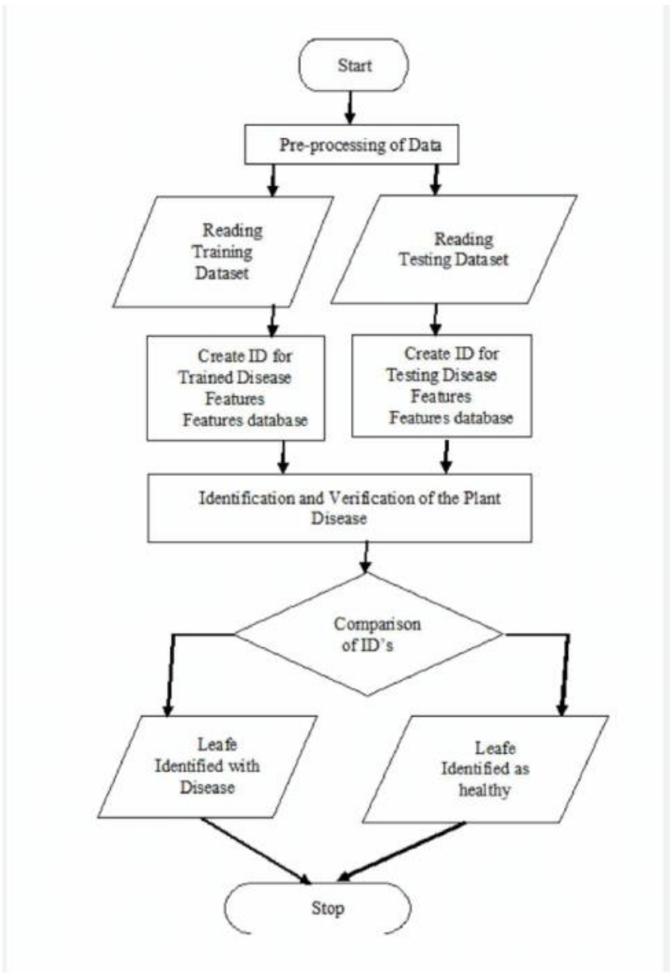
3.Feature Extraction Methods:Explore different feature extraction techniques to capture the relevant information from leaf images. Investigate color-based, texture-based, and shape-based methods to identify the most discriminative features that differentiate healthy leaves from diseased ones. Assess the effectiveness of each technique in representing the unique characteristics of different diseases.

4.Machine Learning Model Selection:Investigate different machine learning algorithms and models suitable for disease detection tasks. Consider convolutional neural networks (CNNs) due to their effectiveness in image classification tasks. Evaluate different architectures, such as VGG, ResNet, or

Inception, to determine the model that yields the best performance in terms of accuracy and efficiency.

5.Performance Metrics:Investigate appropriate performance metrics to evaluate the disease detection system. Common metrics include accuracy, precision, recall, F1-score, and confusion matrix. Analyze these metrics to understand the strengths and limitations of the system and identify areas for improvement.

6.User Interaction and Feedback:Investigate user requirements and conduct user testing to assess the usability and effectiveness of the Flask-based user interface. Gather feedback from farmers or users regarding the ease of uploading images, receiving diagnosis results, and accessing additional information and recommendations. Analyze the feedback to enhance the user experience and address any usability issues.


Start
Pre-processing of Data
Reading Training Dataset
Reading Testing Dataset
Create ID for Trained Disease Features Features database
Create ID for Testing Disease Features Features database
Identification and Verification of the Plant Disease
Comparison of ID's
Leafe Identified with Disease
Leafe Identified as healthy
Stop

5.FLOWCHART

1.Start Program: The program execution begins.

2.Upload Leaf Image: The user uploads an image of a plant leaf through the web interface.

3.Preprocess Image: The uploaded image undergoes preprocessing techniques such as resizing, normalization, and noise removal to enhance its quality and consistency.

4.Disease Detection (Machine Learning): The preprocessed image is inputted into a machine learning model, which detects and classifies the presence of plant diseases based on learned patterns.

5.Disease Diagnosis: The detected diseases are identified, and a diagnosis is made based on the output of the machine learning model. This step may involve mapping the predictions to specific diseases and assessing the severity of the infection.

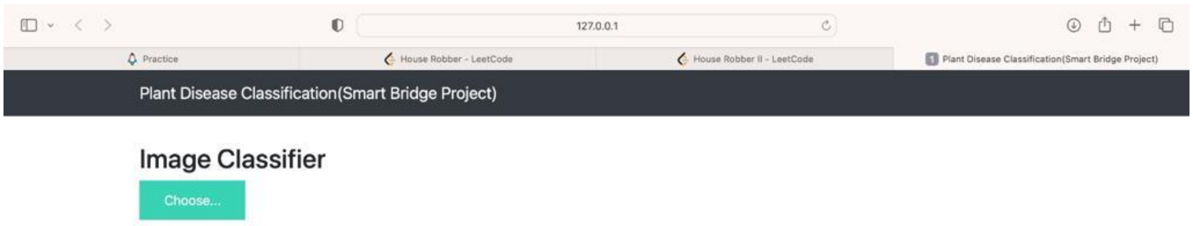
6.Display Results: The diagnosis results are displayed to the user through the web interface, providing information about the detected diseases and their corresponding labels or descriptions.

7.Provide Recommendations: Additional recommendations, such as treatment options, preventive measures, or management strategies, may be provided to the user based on the diagnosed diseases.

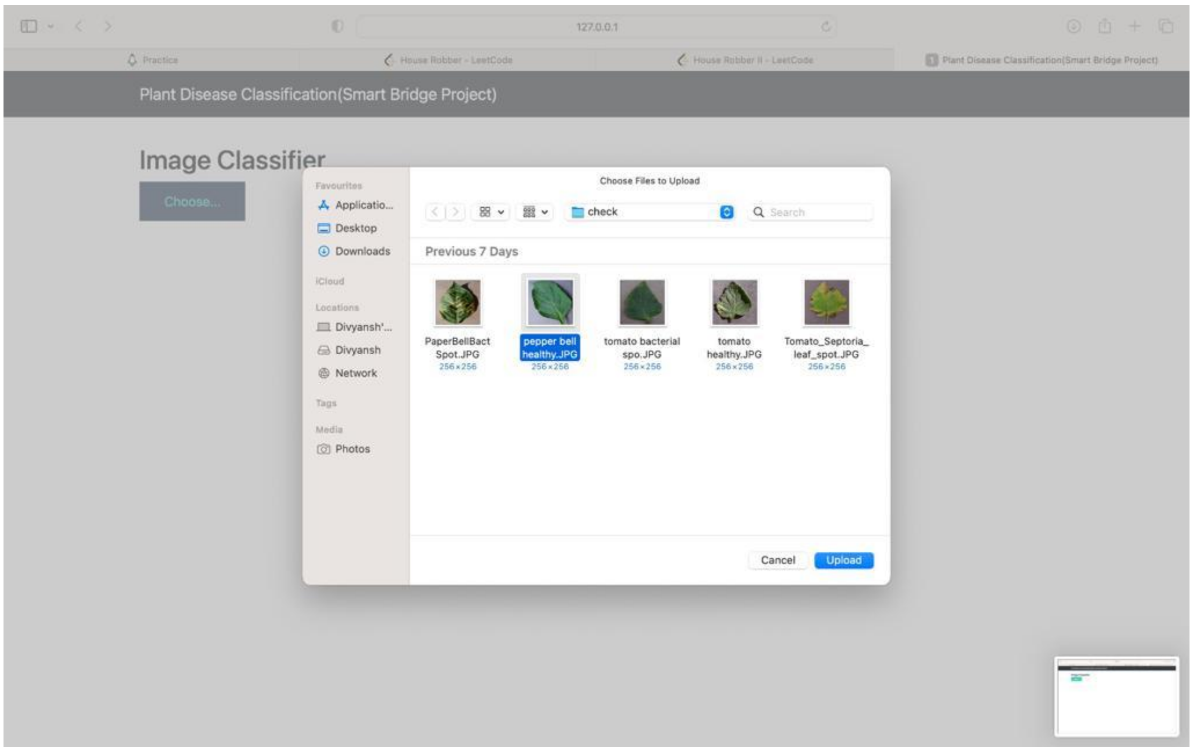
8.End Program: The program execution concludes.

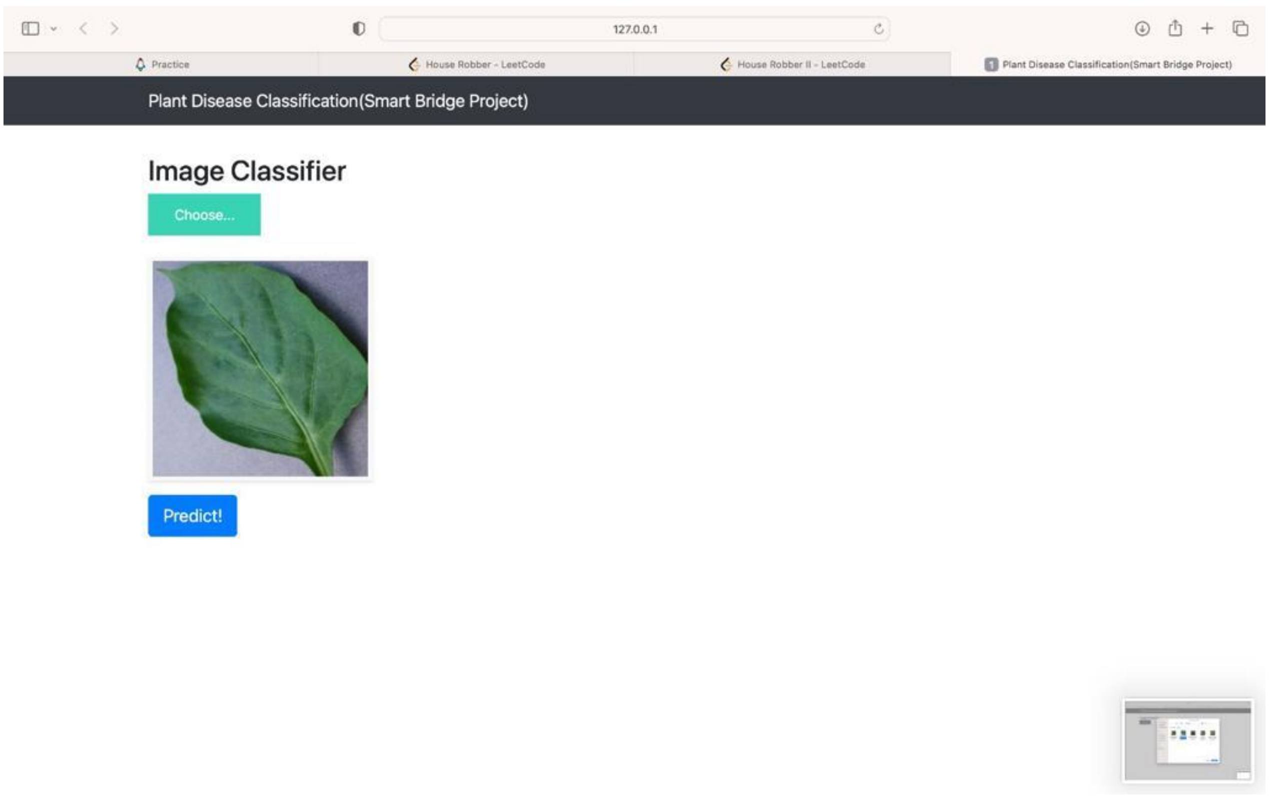
6.RESULTS

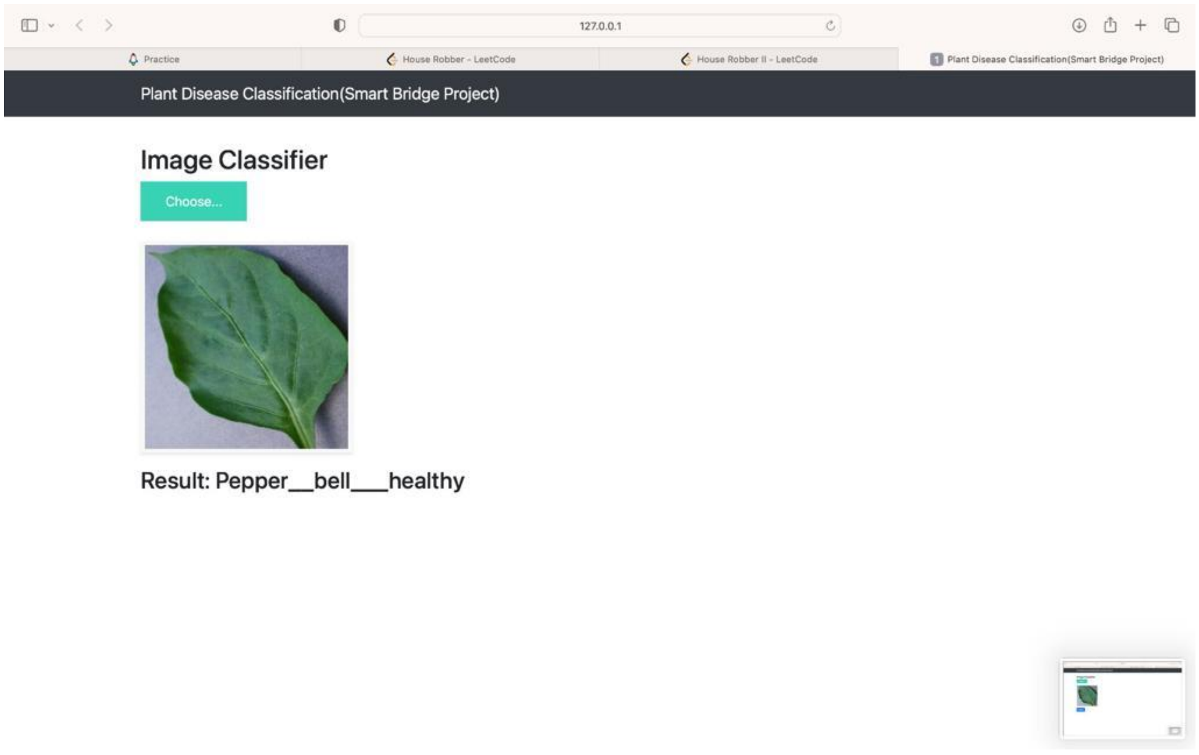
The project's results will be presented and discussed, highlighting the system's effectiveness in accurately detecting and classifying plant diseases. The limitations and potential areas of improvement will be identified, providing insights for future


127.0.0.1
Practice
House Robber - LeetCode
House Robber II - LeetCode
Plant Disease Classification(Smart Bridge Project)
Plant Disease Classification(Smart Bridge Project)
Image Classifier
Choose...

enhancements.




127.0.0.1
Practice
House Robber - LeetCode
House Robber II - LeetCode
Plant Disease Classification(Smart Bridge Project)
Plant Disease Classification(Smart Bridge Project)
Image Classifier
Choose...
Predict!


127.0.0.1
Practice
House Robber - LeetCode
House Robber II - LeetCode
Plant Disease Classification(Smart Bridge Project)
Plant Disease Classification(Smart Bridge Project)
Image Classifier
Choose...

Result: Pepper__bell___healthy

7.ADVANTAGES AND DISADVANTAGES

Advantages:

1.Web-Based Interface: Flask allows for the development of a web-based interface, making the plant disease detection system easily accessible through a web browser. Farmers can access the system from various devices, including smartphones and tablets, without the need for installing additional software.

2.User-Friendly: Flask provides a user-friendly interface that allows farmers to interact with the system intuitively. They can upload leaf images, view disease diagnosis results, and access additional information and recommendations easily. The web-based interface enhances the usability of the system for non-technical users.

3. Flexibility and Customization: Flask is a lightweight and flexible framework, allowing developers to customize and tailor the plant disease detection system to specific requirements. The modular nature of Flask enables the addition of new features, functionalities, and integration with other tools or services as needed.

4.Integration with Python Ecosystem: Flask seamlessly integrates with various Python libraries, tools, and services. This integration enables easy integration with image processing libraries, machine learning frameworks, databases, and external APIs, enhancing the capabilities and functionality of the plant disease detection system

5. Scalability: Flask supports scalable application development, allowing the plant disease detection system to handle increasing workloads and user demands. It facilitates horizontal scaling by deploying multiple instances of the application or vertical scaling by utilizing more powerful server resources.

Disadvantages :

1.Limited Processing Power: Flask is a lightweight framework designed for simplicity and ease of use. However, when dealing with large-scale or computationally intensive tasks, such as processing high-resolution images or training complex machine learning models, the limited processing power of Flask may become a limitation.

2.Deployment and Hosting: While Flask simplifies the deployment process, it still requires setting up a hosting environment, configuring web servers, and managing server infrastructure. This can be challenging for users with limited technical knowledge or for those without access to reliable hosting platforms.

3.Security Considerations: Developing a web-based application using Flask requires careful consideration of security measures. Proper authentication, input validation, and protection against common web vulnerabilities such as cross-site scripting (XSS) and SQL injection are essential to ensure the system's security.

4.Performance: Flask, being a micro-framework, may have limitations in terms of performance and response time, especially when handling a large number of concurrent requests. Careful optimization and efficient resource utilization are necessary to maintain acceptable performance levels.

5.Dependency Management: Flask relies on various Python libraries and dependencies. Managing these dependencies, ensuring compatibility, and keeping them up to date can be challenging, especially as the project evolves or when working with multiple contributors.

8.APPLICATIONS

Plant disease detection using Flask has various applications in agriculture and plant health management. Some key applications include:

1.Early Disease Detection: Plant disease detection systems using Flask can help identify diseases at an early stage, enabling timely intervention and preventing the spread of infections. This allows farmers to take proactive measures such as targeted treatments or preventive actions to minimize crop damage.

2.Precision Agriculture: By accurately detecting and diagnosing plant diseases, Flask-based systems enable farmers to implement precise and targeted agricultural practices. This includes optimizing the use of pesticides, fertilizers, and irrigation resources based on specific disease patterns and severity levels.

3.Disease Management and Control: Plant disease detection systems can provide farmers with insights into effective disease management strategies. Flask-based applications can offer recommendations for disease control measures, including suitable fungicides, cultural practices, and crop rotation techniques, to mitigate disease risks and improve overall plant health.

4.Yield Optimization: By identifying and addressing plant diseases promptly, Flask-based solutions contribute to maximizing crop yields. Early disease detection and appropriate interventions can minimize crop losses, ensuring optimal productivity and reducing economic losses for farmers.

5.Remote Monitoring and Support: Flask-based plant disease detection systems can be accessed remotely, allowing farmers to monitor their crops' health and receive real-time support. This is particularly beneficial in cases where farmers are geographically distant from their fields or require expert advice on disease management.

6.Data Collection and Analysis: Flask-based applications can collect and analyze data on disease occurrences, plant health trends, and geographic distribution of diseases. This data can be used for further research, epidemiological studies, and policy-making to enhance overall plant disease management strategies at regional or national levels.

7.Education and Training: Plant disease detection systems using Flask can serve as educational tools, providing farmers, agronomists, and students with interactive platforms to learn about different plant diseases, their symptoms, and control measures. These applications can contribute to building awareness and knowledge among agricultural communities.

8.Disease Surveillance and Early Warning Systems: Flask-based plant disease detection systems can be integrated into larger disease surveillance networks. By collecting data from multiple sources and analyzing disease patterns, these systems can contribute to the development of early warning systems, helping authorities and stakeholders respond swiftly to disease outbreaks and implement appropriate control measures.

9. CONCLUSION

The proposed plant disease detection system demonstrates the potential of computer vision and machine learning techniques in revolutionizing disease management in agriculture. By providing farmers with a reliable and automated tool for disease identification, the system can significantly reduce crop losses and contribute to sustainable agriculture practices.Overall, plant disease detection using Flask empowers farmers with timely information and actionable insights, leading to improved crop health, reduced losses, and more sustainable agricultural practices. As technology advances and the integration of Flask with other tools and services evolves, the effectiveness and impact of plant disease detection systems are likely to continue to grow, benefiting the agricultural community and global food security.

10. FUTURE SCOPE

The future scope of plant disease detection holds promising advancements and potential developments. Here are some key areas that indicate the future direction of this field:
1.Advanced Machine Learning Techniques: As machine learning and artificial intelligence continue to advance, there is a significant scope for the application of more sophisticated algorithms and models in plant disease detection. Deep learning techniques, such as convolutional neural networks (CNNs), recurrent neural

networks (RNNs), and generative adversarial networks (GANs), can be further explored to improve accuracy and robustness.

2.Edge Computing and IoT Integration: Integrating plant disease detection systems with edge computing and Internet of Things (IoT) devices can enable real-time monitoring and analysis of plants directly in the field. This allows for immediate detection and intervention, reducing response time and enhancing disease management practices.

3.Hyperspectral Imaging: Hyper spectral imaging, which captures images across a wide range of wavelengths, holds great potential for plant disease detection. By analyzing the spectral signatures of plants, hyperspectral imaging can provide detailed information about physiological changes related to diseases, allowing for early detection and precise identification.

10. BIBILOGRAPHY

1. https://medium.com/mlearning-ai/leaf-disease-detection-flask-app-with-source-code-70e46f3f59b7\

2. Saleem, M.H., Potgieter, J. and Arif, K.M., 2019. Plant disease detection and classification by deep learning. *Plants*, *8*(11), p.468.

3 .Ferentinos, K.P., 2018. Deep learning models for plant disease detection and diagnosis. *Computers and electronics in agriculture*, *145*, pp.311-318.

APPENDIX

CODE

```
import tensorflow as tf
                                                                    In [2]:
batchsize=32
imagesize=256
no_classes=15
                                                                   In [14]:
from tensorflow import keras
from tensorflow.keras import layers
                                                                   In [15]:
dataset = tf.keras.preprocessing.image_dataset_from_directory(
    "PlantVillage",
    seed=123,
    shuffle=True,
    image_size=(256,256),
   batch_size=batchsize
)
print(dataset)
```

In [16]:

```
#this dataset has now the images as a tensorflow data
#now the len here says how many batches it has and each bach has 32 images
len(dataset)*batchsize=totalimages
```

In [17]:

```
 pip install tensorflow-macos==2.5

print(len(dataset))

classname=dataset.class_names
print(len(classname))
print(classname)

import numpy
i=None
for image_batch,label_batch in dataset.take(1):
   print(image_batch[0])
   print(label_batch[0].numpy())
   plt.imshow(image_batch[0].numpy().astype("uint8"))#image shape 32 batch size 
256x256 image 3 rgb currently in tensor data will convert it in numpy after wards

   print(classname[label_batch[0]])#label of classes
#[[[1 2 3][]][][[]]]



def partition(ds, train_split=0.8, val_split=0.1, test_split=0.1, shuffle=True, 
shuffle_size=10000):
   assert (train_split + test_split + val_split) == 1

   ds_size = len(ds)

   if shuffle:
      ds = ds.shuffle(shuffle_size, seed=12)

   train_size = int(train_split * ds_size)
   val_size = int(val_split * ds_size)

   train_ds = ds.take(train_size)
   val_ds = ds.skip(train_size).take(val_size)
   test_ds = ds.skip(train_size).skip(val_size)

   return train_ds, val_ds, test_ds
```

In [21]:

```
train_ds, val_ds, test_ds = partition(dataset)
```

In [22]:

```
train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
val_ds = val_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
test_ds = test_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
```

In [23]:

```
resize_and_rescale = tf.keras.Sequential([
  layers.experimental.preprocessing.Resizing(256, 256),
  layers.experimental.preprocessing.Rescaling(1./255),
])
```

In [24]:

```
data_augmentation = tf.keras.Sequential([
  layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical"),
  layers.experimental.preprocessing.RandomRotation(0.2),
])
```

In [28]:

```
from tensorflow.keras import models
input__shape = (32, 256, 256, 3)
n_classes = 15

cnn = models.Sequential([
    resize_and_rescale,#data_augmentation
    layers.Conv2D(filters=32, kernel_size = (3,3), activation='relu',
input_shape=input__shape),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64,  kernel_size = (3,3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64,  kernel_size = (3,3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(n_classes, activation='softmax'),
])

cnn.build(input_shape=input__shape)
```

In [29]:

```
cnn.summary()


cnn.compile(
    optimizer='adam',
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    metrics=['accuracy']
)
```

In [31]:

```
history = cnn.fit(
   train_ds,
   batch_size=32,
   validation_data=val_ds,
   verbose=1,
   epochs=10,
)

score=cnn.evaluate(test_ds)
```

In [ ]:

```
print(score)
```

In [ ]:

In [ ]:

```
import numpy as np
for images_batch, labels_batch in test_ds.take(1):

   first_image = images_batch[0].numpy().astype('uint8')
   first_label = labels_batch[0].numpy()

   print("first image to predict")
   plt.imshow(first_image)
   print("actual label:",classname[first_label])

   batch_prediction = cnn.predict(images_batch)


   print("predicted label:",classname[np.argmax(batch_prediction[0])])
   print("Confidence Level=",max(batch_prediction[0])*100)
   print()
   print(batch_prediction[0])
```

In [32]:

```
from keras.models import load_model
model = load_model('2st.h5')
model.summary()


import numpy as np
image = tf.keras.utils.load_img(path='testing/1.jpg',target_size=(256,256))
input_arr = tf.keras.utils.img_to_array(image)
input_arr = np.array([input_arr])  # Convert single image to a batch.
```

In [18]:

```
input_arr[0]


output=model.predict(input_arr)
print(output)
```

```
classname=['Pepper__bell___Bacterial_spot', 'Pepper__bell___healthy',
'Potato___Early_blight', 'Potato___Late_blight', 'Potato___hea

print("predicted label:",classname[np.argmax(output[0])])
plt.imshow(input_arr[0].astype("uint8"))
```