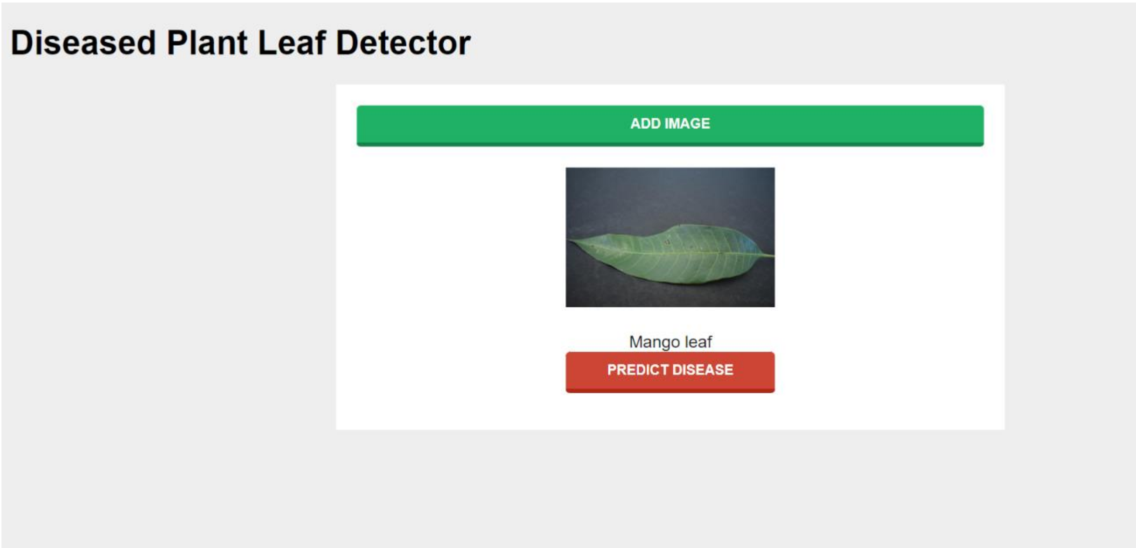
Diseased Plant Leaf Detector

Name : M.M.Chathuni Devindi Manage

University : IIT



Content

1. Introduction

This project aims to develop a plant disease detection model utilizing machine learning techniques. Timely detection of plant diseases is crucial for effective agricultural management and sustaining crop yields. Automating this process through machine learning can significantly enhance efficiency, accuracy, and overall agricultural productivity.

2. Dataset

The dataset used for this project contains images of plant leaves affected by different diseases. The dataset consists of three main categories: Mango, Lemon, and Guava. Each category represents a specific type of plant. Images in these categories were preprocessed, resized to 256x256 pixels, and normalized to facilitate training.

https://drive.google.com/drive/folders/19TP77E_e0UakueReC5pbFhoE-1quIKds?usp=sharing

3. Model Architecture

The designed model follows a Convolutional Neural Network (CNN) architecture. The model comprises multiple layers, including convolutional layers, max-pooling layers, and dense layers.

Input (256x256x3)

(Convolutional Layer 1)

Conv2D (32 filters, 3x3 kernel, ReLU)

(MaxPooling Layer 1)

MaxPooling2D (3x3 pool size)

(Convolutional Layer 2)

Conv2D (16 filters, 3x3 kernel, ReLU)

(MaxPooling Layer 2)

MaxPooling2D (2x2 pool size)

(Flatten Layer)

Flatten

(Dense Layer 1)

Dense (8 neurons, ReLU)

(Dense Layer 2 - Output)

Dense (3 neurons, Softmax)

4. Training

The dataset was split into training, validation, and test sets. The model was compiled using categorical cross-entropy loss and the Adam optimizer. The training process consisted of 50 epochs with a batch size of 128. The training set was further split for validation to monitor model performance during training.

5. Explain the Code

- **Mounting Google Drive and Accessing Dataset**

This code snippet mounts Google Drive to access files and datasets stored in your Google Drive account. The **ls** command lists the contents of the specified directory in Google Drive.

```
[ ] from google.colab import drive
    drive.mount('/content/drive')

    Mounted at /content/drive

!ls'/content/drive/My Drive/InternFinalProject/Plant-Leaf-Disease-Predictor'

    /bin/bash: line 1: ls/content/drive/My Drive/InternFinalProject/Plant-Leaf-Disease-Predictor:

[ ] !ls"/content/drive/My Drive/InternFinalProject/Plant-Leaf-Disease-Predictor/dataset"

    /bin/bash: line 1: ls/content/drive/My Drive/InternFinalProject/Plant-Leaf-Disease-Predictor/d
```

- **Importing Libraries**

Several Python libraries are imported to assist in building and training the plant disease detection model:

NumPy	: For numerical operations.
Pandas	: For data manipulation and analysis.
Matplotlib	: For creating visualizations.
cv2 (OpenCV)	: For image processing.
Random	: For generating random numbers.
OS	: For interacting with the operating system.
PIL (Pillow)	: For image processing tasks.
TensorFlow	: An open-source machine learning framework.
Keras	: A high-level neural networks API running on top of TensorFlow.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import cv2
import random
import os
from os import listdir
from PIL import Image
import tensorflow as tf
from keras.preprocessing import image
#from tensorflow.keras.utils import img_to_array, array_to_img
from keras.optimizers import Adam
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D
#from keras.layers import activation, Flatten, Dropout, Dense
from sklearn.model_selection import train_test_split
from keras.models import model_from_json
from keras.utils import to_categorical
```

```
!pip install activation
```

```
Collecting activation
  Downloading activation-0.1.0.tar.gz (13 kB)
  Preparing metadata (setup.py) ... done
Requirement already satisfied: Click>=6.0 in /usr/local/lib/python3.10/dist-packages (from acti
Building wheels for collected packages: activation
  Building wheel for activation (setup.py) ... done
  Created wheel for activation: filename=activation-0.1.0-py2.py3-none-any.whl size=4072 sha25
  Stored in directory: /root/.cache/pip/wheels/4a/38/01/6f221ac293945be47e90ddac9ee671f5698d1d
Successfully built activation
Installing collected packages: activation
Successfully installed activation-0.1.0
```

```
from keras.activations import relu
```

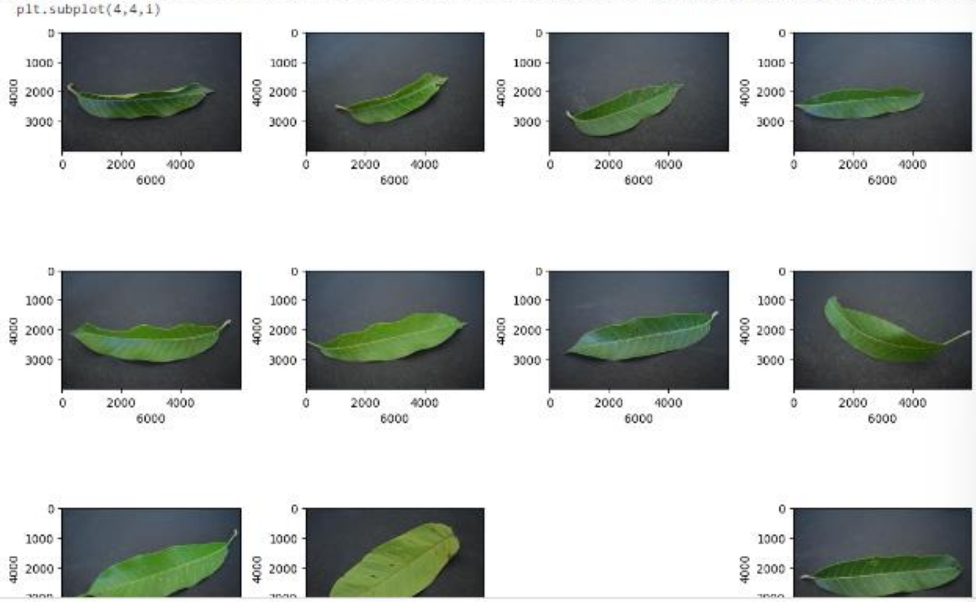
```
#from keras.layers import activation
from keras.layers import Flatten
from keras.layers import Dropout
from keras.layers import Dense
```

- **Plotting Sample Images from the Dataset**

This snippet displays 16 random images from the 'Mango' dataset to visually inspect the data.

```
#plotting 12 images to check dataset
plt.figure(figsize=(12,12))
path="/content/drive/My Drive/InternFinalProject/Plant-Leaf-Disease-Predictor/dataset/Mango"
for i in range(1,17):
  plt.subplot(4,4,i)
  plt.tight_layout()
  rand_img=plt.imread(path+'/'+random.choice(sorted(os.listdir(path))))
  plt.imshow(rand_img)
  plt.xlabel(rand_img.shape[1], fontsize=10)
  plt.ylabel(rand_img.shape[0], fontsize=10)
```

```
<ipython-input-10-b9e5e79245ef>:5: MatplotlibDeprecationWarning: Auto-removal of overlapping axes is deprecated since 3.6
  plt.subplot(4,4,i)
```



- **Data Preprocessing**

The code includes functions for converting images to arrays, loading and preprocessing the dataset, and splitting it into training and testing sets.

```
[ ] def convert_image_to_array (image_dir):
      try:
        image=cv2.imread(image_dir)
        if image is not None :
          image=cv2.resize(image, (256,256))
          return img_to_array(image)
        else:
          return np.array([])
      except Exception as e:
        print(f"Error : {e}")
        return None
```

```
[ ] dir="/content/drive/My Drive/InternFinalProject/Plant-Leaf-Disease-Predictor/dataset"
    image_list, label_list=[],[]
    all_labels=['Mango','Lemon','Guava']
    binary_labels=[0,1,2]
    temp=-1

    for directory in['Mango', 'Lemon','Guava']:
      plant_image_list=listdir(f"{dir}/{directory}")
      temp+=1
      for files in plant_image_list:
        image_path=f"{dir}/{directory}/{files}"
        image_list.append(convert_image_to_array(image_path))
        label_list.append(binary_labels[temp])
```

check for dataset imbalance or not

```
[ ] label_counts=pd.DataFrame(label_list).value_counts()
    label_counts.head()
```

```
0    236
1    236
2    236
dtype: int64
```

splitting dataset into train and test

```
[ ] x_train, x_test, y_train, y_test=train_test_split (image_list, label_list, test_size=0.2, random_state=10)
```

```
[ ] x_train=np.array(x_train, dtype=np.float16)/255.0
    x_test=np.array(x_test, dtype=np.float16)/255.0
    x_train=x_train.reshape(-1,256,256,3)
    x_test=x_test.reshape(-1,256,256,3)
```

one-hot encoding

```
[ ] y_train=to_categorical(y_train,num_classes=3)
    y_test=to_categorical(y_test,num_classes=3)
```

- **Model Architecture**

This section defines the architecture of the convolutional neural network (CNN) model using Keras. It consists of convolutional layers, max-pooling layers, a flatten layer, and dense layers.

```
[ ] model=Sequential()
    model.add(Conv2D(32,(3,3),padding="same", input_shape=(256,256,3), activation="relu"))
    model.add(MaxPooling2D(pool_size=(3,3)))
    model.add(Conv2D(16,(3,3),padding="same",  activation="relu"))
    model.add(MaxPooling2D(pool_size=(2,2)))
    model.add(Flatten())
    model.add(Dense(8, activation="relu"))
    model.add(Dense(3, activation="softmax"))
    model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 256, 256, 32)      896

 max_pooling2d (MaxPooling2  (None, 85, 85, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 85, 85, 16)        4624

 max_pooling2d_1 (MaxPoolin  (None, 42, 42, 16)        0
 g2D)

 flatten (Flatten)           (None, 28224)             0

 dense (Dense)               (None, 8)                 225800

 dense_1 (Dense)             (None, 3)                 27

=================================================================
Total params: 231347 (903.70 KB)
Trainable params: 231347 (903.70 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

compile the model

```
[ ] model.compile(loss='categorical_crossentropy', optimizer=Adam(0.001), metrics=['accuracy'])
```

- **Training**

The model is trained using the training data and validated using a portion of the training set.

split train set into training and validation set

```
[ ] x_train, x_val, y_train, y_val=train_test_split (x_train, y_train, test_size=0.2, random_state=10)
```

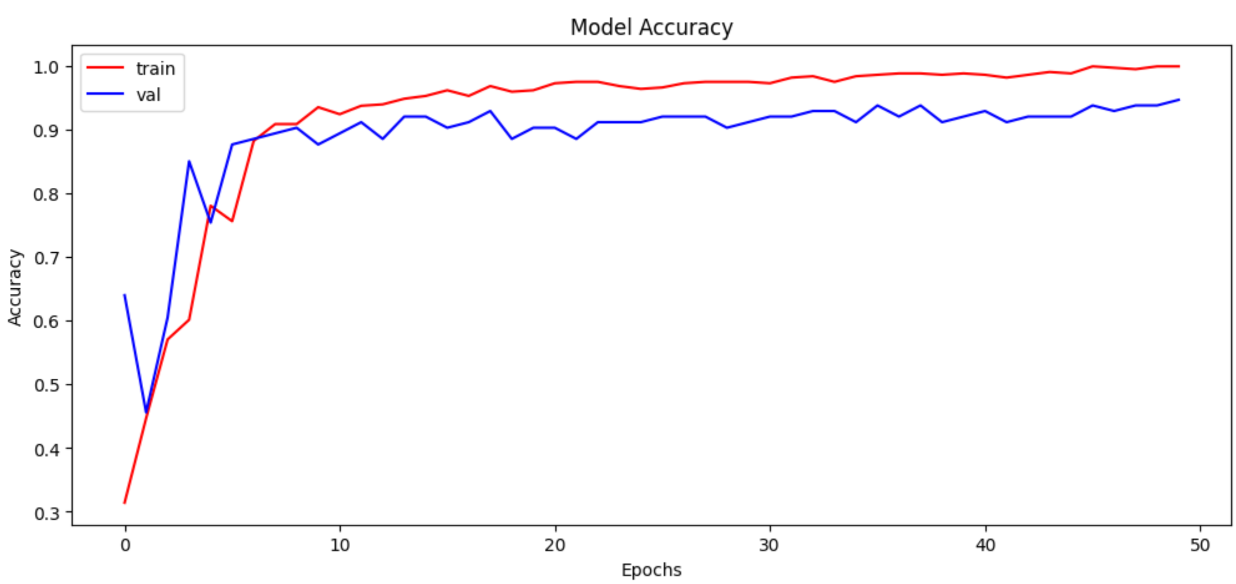
Train the model

```
[ ] from sklearn.utils import validation
    epochs=50
    batch_size=128
    history=model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, validation_data=(x_val,y_val))
```

```
Epoch 1/50
4/4 [==============================] - 29s 7s/step - loss: 1.0939 - accuracy: 0.3142 - val_loss: 1.0798 - v
Epoch 2/50
4/4 [==============================] - 35s 9s/step - loss: 1.0625 - accuracy: 0.4469 - val_loss: 0.9883 - v
Epoch 3/50
4/4 [==============================] - 29s 6s/step - loss: 1.0079 - accuracy: 0.5708 - val_loss: 0.9288 - v
Epoch 4/50
4/4 [==============================] - 28s 7s/step - loss: 0.9330 - accuracy: 0.6018 - val_loss: 0.8232 - v
Epoch 5/50
4/4 [==============================] - 27s 7s/step - loss: 0.8067 - accuracy: 0.7810 - val_loss: 0.7281 - v
Epoch 6/50
4/4 [==============================] - 25s 6s/step - loss: 0.6767 - accuracy: 0.7566 - val_loss: 0.5917 - v
Epoch 7/50
4/4 [==============================] - 26s 6s/step - loss: 0.5428 - accuracy: 0.8827 - val_loss: 0.4892 - v
Epoch 8/50
4/4 [==============================] - 28s 7s/step - loss: 0.4276 - accuracy: 0.9093 - val_loss: 0.4288 - v
Epoch 9/50
4/4 [==============================] - 25s 7s/step - loss: 0.3709 - accuracy: 0.9093 - val_loss: 0.3594 - v
Epoch 10/50
4/4 [==============================] - 28s 7s/step - loss: 0.2900 - accuracy: 0.9358 - val_loss: 0.2986 - v
Epoch 11/50
4/4 [==============================] - 27s 6s/step - loss: 0.2625 - accuracy: 0.9248 - val_loss: 0.2631 - v
Epoch 12/50
4/4 [==============================] - 27s 6s/step - loss: 0.2391 - accuracy: 0.9381 - val_loss: 0.2614 - v
Epoch 13/50
4/4 [==============================] - 27s 7s/step - loss: 0.2139 - accuracy: 0.9403 - val_loss: 0.2322 - v
Epoch 14/50
4/4 [==============================] - 29s 8s/step - loss: 0.1913 - accuracy: 0.9491 - val_loss: 0.2338 - v
Epoch 15/50
4/4 [==============================] - 27s 7s/step - loss: 0.1654 - accuracy: 0.9535 - val_loss: 0.2008 - v
Epoch 16/50
4/4 [==============================] - 27s 7s/step - loss: 0.1573 - accuracy: 0.9624 - val_loss: 0.2046 - v
Epoch 17/50
4/4 [==============================] - 27s 6s/step - loss: 0.1497 - accuracy: 0.9535 - val_loss: 0.1850 - v
Epoch 18/50
```

- **Plot Accuracy**

```
[ ] plt.figure(figsize=(12,5))
    plt.plot(history.history['accuracy'],color='r')
    plt.plot(history.history['val_accuracy'], color='b')
    plt.title('Model Accuracy')
    plt.ylabel('Accuracy')
    plt.xlabel('Epochs')
    plt.legend(['train','val'])
    plt.show
```



- **Model Evaluation**

The trained model is evaluated on the test set, and the accuracy is printed.

```
[ ] print("Calculating model accuracy")
    scores=model.evaluate(x_test, y_test)
    print(f"Test Accuracy:{scores[1]*100}")

    Calculating model accuracy
    5/5 [==============================] - 2s 404ms/step - loss: 0.0630 - accuracy: 0.9789
    Test Accuracy:97.88732528686523
```

- **Model Prediction**

This code generates predictions for the test set using the trained model.

```
[ ] y_pred=model.predict(x_test)

    5/5 [==============================] - 2s 414ms/step
```

- **Result Visualization**

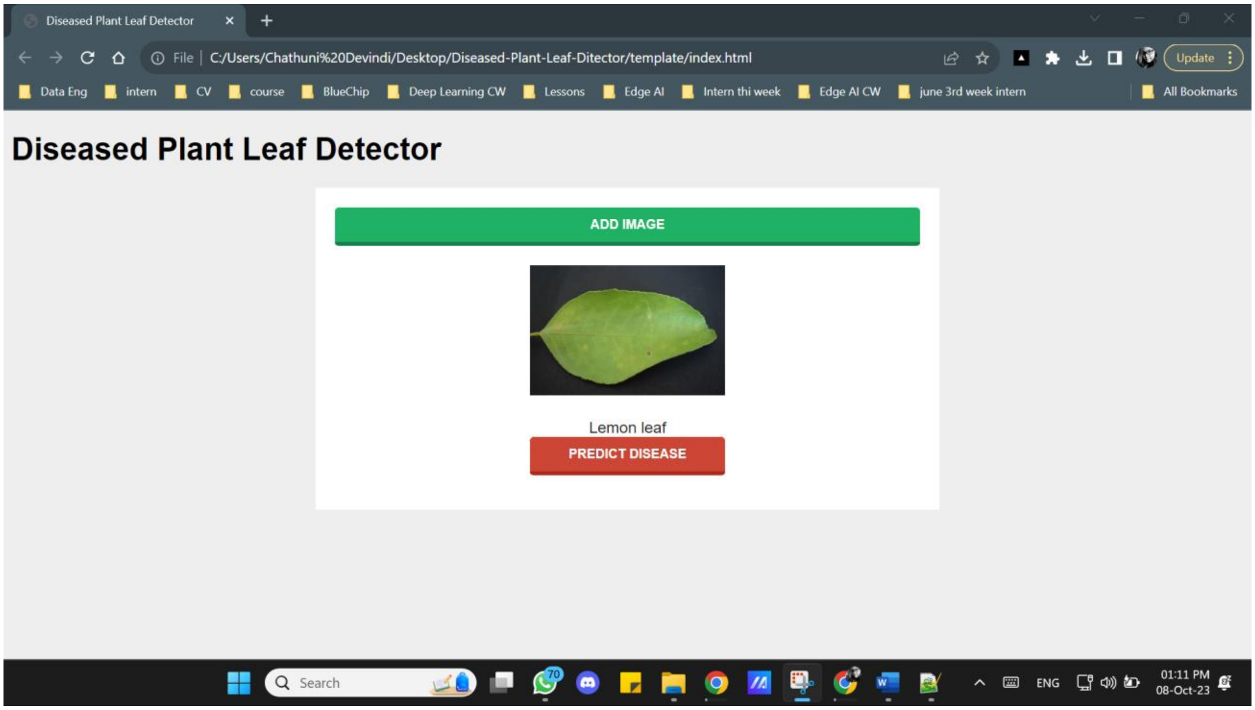
```
[ ] from keras.src.utils import array_to_img
    #plotting image to compare
    img=array_to_img(x_test[11])
    img
```

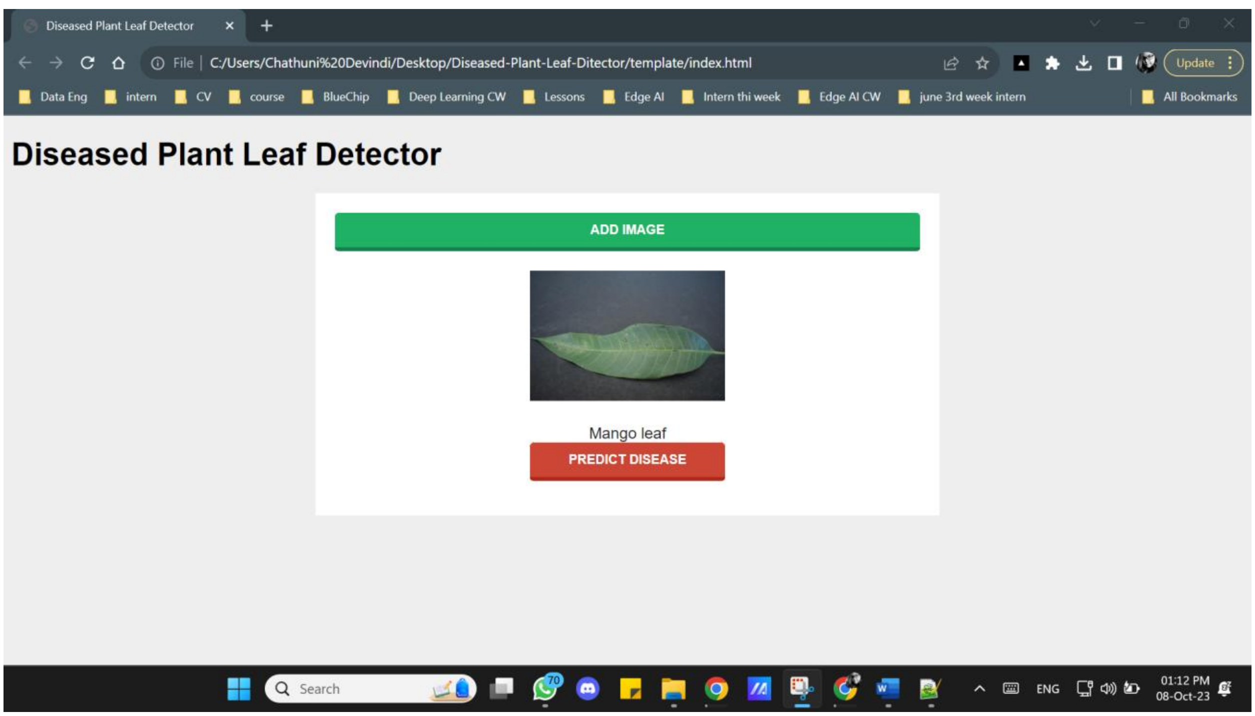
```
[ ] print("Orginal Label: ", all_labels[np.argmax(y_test[11])])
    print("Predicted Label: ", all_labels[np.argmax(y_pred[4])])
    print(y_pred[2])

    Orginal Label:  Mango
    Predicted Label:  Guava
    [1.4160594e-06 9.9996716e-01 3.1407315e-05]
```

```
[ ] for i in range(50):
      print(all_labels[np.argmax(y_test[i])], "-", all_labels[np.argmax(y_pred[i])])
```

```
Guava - Guava
Mango - Mango
Lemon - Lemon
Lemon - Lemon
Guava - Guava
Lemon - Lemon
Mango - Mango
Mango - Mango
Guava - Guava
Lemon - Lemon
Guava - Guava
Mango - Mango
Lemon - Lemon
Mango - Mango
Lemon - Lemon
Guava - Guava
Guava - Guava
Guava - Guava
Lemon - Lemon
Guava - Guava
Mango - Mango
Lemon - Lemon
Lemon - Lemon
Mango - Guava
Lemon - Lemon
Mango - Mango
Guava - Guava
Lemon - Lemon
Mango - Mango
Guava - Guava
Mango - Mango
Guava - Guava
Lemon - Lemon
Mango - Mango
```

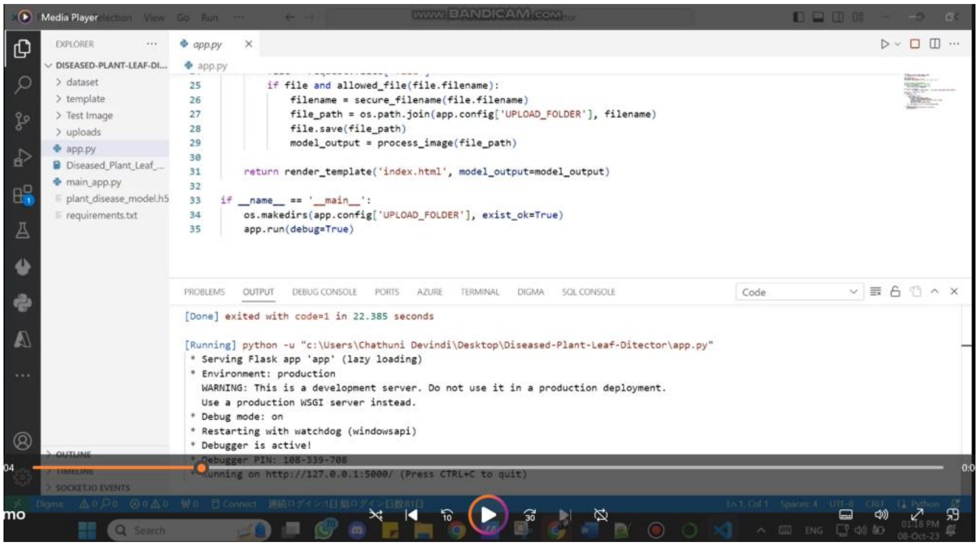
6. Interface of the Application





7. Demo

Demo.mp4 https://drive.google.com/file/d/1cMJQR8GLvt4O9t9sbq0PjF74Xj7iiU3H/view?usp=sharing



8. Conclusion

The project successfully achieved its objective of developing a diseased plant detection model using machine learning. The trained model demonstrated promising accuracy in identifying diseases from plant leaf images. The automation of disease detection in plants holds great potential for improving agricultural practices, minimizing economic losses, and contributing to sustainable and efficient farming methods.