**ABSTRACT:**

In modern agriculture, early detection and accurate diagnosis of plant diseases are critical for preventing crop loss and ensuring food security. Leveraging advanced machine learning techniques, this project develops a plant disease prediction system using Artificial Neural Networks (ANNs). By implementing a convolutional neural network (CNN), a specialized type of ANN, the system is designed to analyze plant images and classify them according to the presence of specific diseases. The CNN is trained on a diverse dataset of labeled images, allowing it to learn and identify disease-specific patterns effectively. The model’s ability to predict diseases from new images demonstrates its potential to significantly enhance agricultural practices by facilitating timely interventions. This approach highlights the transformative impact of machine learning on plant health management and agricultural efficiency.

**INTRODUCTION:**

Plant diseases are a major challenge in agriculture, posing threats to crop health, productivity, and global food security. As agricultural practices become more intensive, traditional methods of diagnosing plant diseases—primarily relying on visual inspection and expert knowledge—are proving insufficient. These methods are often time-consuming, labor-intensive, and subject to variability in human judgment. This underscores the need for innovative solutions that can provide more efficient, accurate and scalable disease detection.

Advancements in machine learning and artificial intelligence (AI) offer promising alternatives to conventional diagnostic approaches. Among these technologies, Convolutional Neural Networks (CNNs), a specialized type of Artificial Neural Network (ANN), have emerged as particularly effective for image recognition tasks. CNNs excel at automatically extracting and learning features from visual data, making them well-suited for analyzing plant images to identify disease symptoms and patterns.

This project leverages CNNs to develop a plant disease prediction system that uses image analysis to diagnose various plant diseases. By training the model on a diverse dataset of labeled plant images, the system learns to recognize distinctive features associated with different diseases. Once trained, the model can predict the presence of diseases from new images, providing a powerful tool for early detection and intervention.

The implementation of this system through a user-friendly interface, created with Gradio, enables users to easily upload plant images and receive accurate disease predictions. This approach not only streamlines the diagnostic process but also enhances accessibility for farmers and agricultural professionals, promoting more effective plant health management. Ultimately, the project demonstrates the transformative potential of AI in agriculture, paving the way for more efficient and reliable disease management solutions.

* Plant diseases are a critical challenge in agriculture, impacting crop yield and food security. Traditional diagnostic methods are often slow and subjective.
* The project utilizes CNNs, a type of ANN, to leverage image recognition capabilities for disease detection. CNNs are known for their efficacy in analyzing visual data and identifying intricate patterns.
* The system involves training a CNN on a diverse dataset of labelled plant images. This process enables the model to learn and recognize disease-specific features and patterns.
* The trained model can accurately classify new plant images, providing valuable insights for early disease detection and intervention. This approach aims to enhance agricultural practices by facilitating faster and more reliable disease diagnosis.
* The project highlights the potential of machine learning in revolutionizing plant disease management and suggests future work to expand the dataset, improve model performance, and integrate the system into practical agricultural tools.

By integrating these modern computational techniques, the project aims to provide a robust solution for plant disease prediction, demonstrating the transformative impact of AI on agricultural diagnostics and crop management.

**SPECIFICATION:**

**Hardware Requirements:**

1. **Development and Training Phase:**

* **CPU**: High-performance multi-core (e.g., Intel i7, AMD Ryzen 7), optionally more powerful (e.g., Intel i9, AMD Ryzen 9)
* **GPU**: Dedicated NVIDIA GPU (e.g., RTX 3080, RTX 3090, A100 Tensor Core) for accelerated training
* **RAM**: Minimum 16 GB, ideally 32 GB for handling large datasets
* **Storage**: SSD with at least 1 TB capacity for fast data access
* **Cooling**: Efficient system cooling (e.g., case fans, liquid cooling) to manage heat

1. **Deployment Phase:**

* **CPU**: Capable processor (e.g., Intel i5 or ARM equivalent), optionally similar to training phase CPU
* **GPU**: Optional, depending on real-time or high-throughput needs (e.g., NVIDIA GPUs)
* **RAM**: 8 GB to 16 GB for servers or cloud, lower for edge devices
* **Storage**: Adequate storage for model and data, scalable for cloud-based solutions
* **Networking**: Reliable network infrastructure for cloud or robust local network for on-site deployments

**Software Requirements:**

1. **Programming Language:**

* **Python**: Primary language for developing machine learning models and implementing the prediction system due to its extensive libraries and frameworks for data science and AI.

1. **Machine Learning Frameworks:**

* **TensorFlow** and **Keras**: For building and training Convolutional Neural Networks (CNNs) and other deep learning models.
* **PyTorch**: An alternative deep learning framework, if used, for model development and training.

1. **Data Processing Libraries:**

* **NumPy**: For numerical operations and handling large datasets.
* **Pandas**: For data manipulation and analysis, especially useful for preprocessing the image data.
* **OpenCV**: For image processing tasks, such as resizing and augmentation.

1. **Frontend Development:**

* **Gradio:** For creating interactive web interfaces that allow users to upload plant images and receive disease predictions from the model.

1. **Backend Development:**

* **Django**: Framework for building web services to serve the model if needed for integration with a web-based application.

1. **Development Environment:**

* **Google Colaboratory**: Colab is a hosted Jupyter Notebook service that requires no setup to use and provides free access to computing resources, including GPUs and TPUs. Colab is especially well suited to machine learning, data science, and education.

**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)

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

!pip install kaggle

kaggle\_credentails = json.load(open("kaggle.json"))

# setup Kaggle API key as environment variables

os.environ['KAGGLE\_USERNAME'] = kaggle\_credentails["username"]

os.environ['KAGGLE\_KEY'] = kaggle\_credentails["key"]

!kaggle datasets download -d abdallahalidev/plantvillage-dataset

!ls

# Unzip the downloaded dataset

with ZipFile("plantvillage-dataset.zip", 'r') as zip\_ref:

    zip\_ref.extractall()

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])

print(len(os.listdir("plantvillage dataset/color/Grape\_\_\_healthy")))

print(os.listdir("plantvillage dataset/color/Grape\_\_\_healthy")[:5])

# Dataset Path

base\_dir = 'plantvillage dataset/color'

image\_path = '/content/plantvillage dataset/color/Apple\_\_\_Cedar\_apple\_rust/025b2b9a-0ec4-4132-96ac-7f2832d0db4a\_\_\_FREC\_C.Rust 3655.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()

image\_path = '/content/plantvillage dataset/color/Apple\_\_\_Cedar\_apple\_rust/025b2b9a-0ec4-4132-96ac-7f2832d0db4a\_\_\_FREC\_C.Rust 3655.JPG'

# Read the image

img = mpimg.imread(image\_path)

print(img)

# Image Parameters

img\_size = 224

batch\_size = 32

# Image Data Generators

data\_gen = ImageDataGenerator(

    rescale=1./255,

    validation\_split=0.2  # Use 20% of data for validation

)

# Train Generator

train\_generator = data\_gen.flow\_from\_directory(

    base\_dir,

    target\_size=(img\_size, img\_size),

    batch\_size=batch\_size,

    subset='training',

    class\_mode='categorical'

)

# Validation Generator

validation\_generator = data\_gen.flow\_from\_directory(

    base\_dir,

    target\_size=(img\_size, img\_size),

    batch\_size=batch\_size,

    subset='validation',

    class\_mode='categorical'

)

# Model Definition

model = models.Sequential()

model.add(layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(img\_size, img\_size, 3)))

model.add(layers.MaxPooling2D(2, 2))

model.add(layers.Conv2D(64, (3, 3), activation='relu'))

model.add(layers.MaxPooling2D(2, 2))

model.add(layers.Flatten())

model.add(layers.Dense(256, activation='relu'))

model.add(layers.Dense(train\_generator.num\_classes, activation='softmax'))

# Compile the Model

model.compile(optimizer='adam',

              loss='categorical\_crossentropy',

              metrics=['accuracy'])

# Training the Model

history = model.fit(

    train\_generator,

    steps\_per\_epoch=train\_generator.samples // batch\_size,  # Number of steps per epoch

    epochs=5,  # Number of epochs

    validation\_data=validation\_generator,

    validation\_steps=validation\_generator.samples // batch\_size  # Validation steps

)

# Model Evaluation

print("Evaluating model...")

val\_loss, val\_accuracy = model.evaluate(validation\_generator, steps=validation\_generator.samples // batch\_size)

print(f"Validation Accuracy: {val\_accuracy \* 100:.2f}%")

# Plot training & validation accuracy values

plt.plot(history.history['accuracy'])

plt.plot(history.history['val\_accuracy'])

plt.title('Model accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend(['Train', 'Test'], loc='upper left')

plt.show()

# Plot training & validation loss values

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.title('Model loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend(['Train', 'Test'], loc='upper left')

plt.show()

# Function to Load and Preprocess the Image using Pillow

def load\_and\_preprocess\_image(image\_path, target\_size=(224, 224)):

    # Load the image

    img = Image.open(image\_path)

    # Resize the image

    img = img.resize(target\_size)

    # Convert the image to a numpy array

    img\_array = np.array(img)

    # Add batch dimension

    img\_array = np.expand\_dims(img\_array, axis=0)

    # Scale the image values to [0, 1]

    img\_array = img\_array.astype('float32') / 255.

    return img\_array

# Function to Predict the Class of an Image

def predict\_image\_class(model, image\_path, class\_indices):

    preprocessed\_img = load\_and\_preprocess\_image(image\_path)

    predictions = model.predict(preprocessed\_img)

    predicted\_class\_index = np.argmax(predictions, axis=1)[0]

    predicted\_class\_name = class\_indices[predicted\_class\_index]

    return predicted\_class\_name

# Create a mapping from class indices to class names

class\_indices = {v: k for k, v in train\_generator.class\_indices.items()}

class\_indices

# saving the class names as json file

json.dump(class\_indices, open('class\_indices.json', 'w'))

# Example Usage

image\_path = '/content/test\_apple\_black\_rot.jpg'

#image\_path = '/content/test\_blueberry\_healthy.jpg'

#image\_path = '/content/test\_potato\_early\_blight.jpg'

predicted\_class\_name = predict\_image\_class(model, image\_path, class\_indices)

# Output the result

print("Predicted Class Name:", predicted\_class\_name)

model.save('drive/MyDrive/Youtube/trained\_models/plant\_disease\_prediction\_model.h5')

import gradio as gr

# Function to load and preprocess image

def load\_and\_preprocess\_image(image):

    # Resize the image

    img = image.resize((224, 224))

    # Convert the image to a numpy array

    img\_array = np.array(img)

    # Add batch dimension

    img\_array = np.expand\_dims(img\_array, axis=0)

    # Scale the image values to [0, 1]

    img\_array = img\_array.astype('float32') / 255.

    return img\_array

# Function to predict the class of an image

def predict\_image\_class(image):

    preprocessed\_img = load\_and\_preprocess\_image(image)

    predictions = model.predict(preprocessed\_img)

    predicted\_class\_index = np.argmax(predictions, axis=1)[0]

    predicted\_class\_name = class\_indices[predicted\_class\_index]

    return predicted\_class\_name

# Create Gradio interface

iface = gr.Interface(

    fn=predict\_image\_class,

    inputs=gr.Image(type="pil"),

    outputs="text",

    title="Plant Disease Prediction - Using ANN ",

    description="Upload an image of a plant leaf to predict its disease."

)

# Launch the interface

iface.launch()

**RESULTS AND DISCUSSIONS:**

The results of the plant disease prediction system indicate strong performance and reliability. The Convolutional Neural Network (CNN) model achieved high accuracy in identifying various plant diseases, demonstrating its effectiveness in classifying plant images correctly. Evaluation metrics such as precision, recall, and F1-score confirmed that the model performs well across different disease categories, though some challenges were observed with certain diseases. The Gradio interface facilitated easy interaction, with fast response times for processing user-submitted images. Overall, the system provides a user-friendly and efficient tool for disease detection, outperforming traditional diagnostic methods in terms of speed and accuracy, and offering valuable support for plant health management.

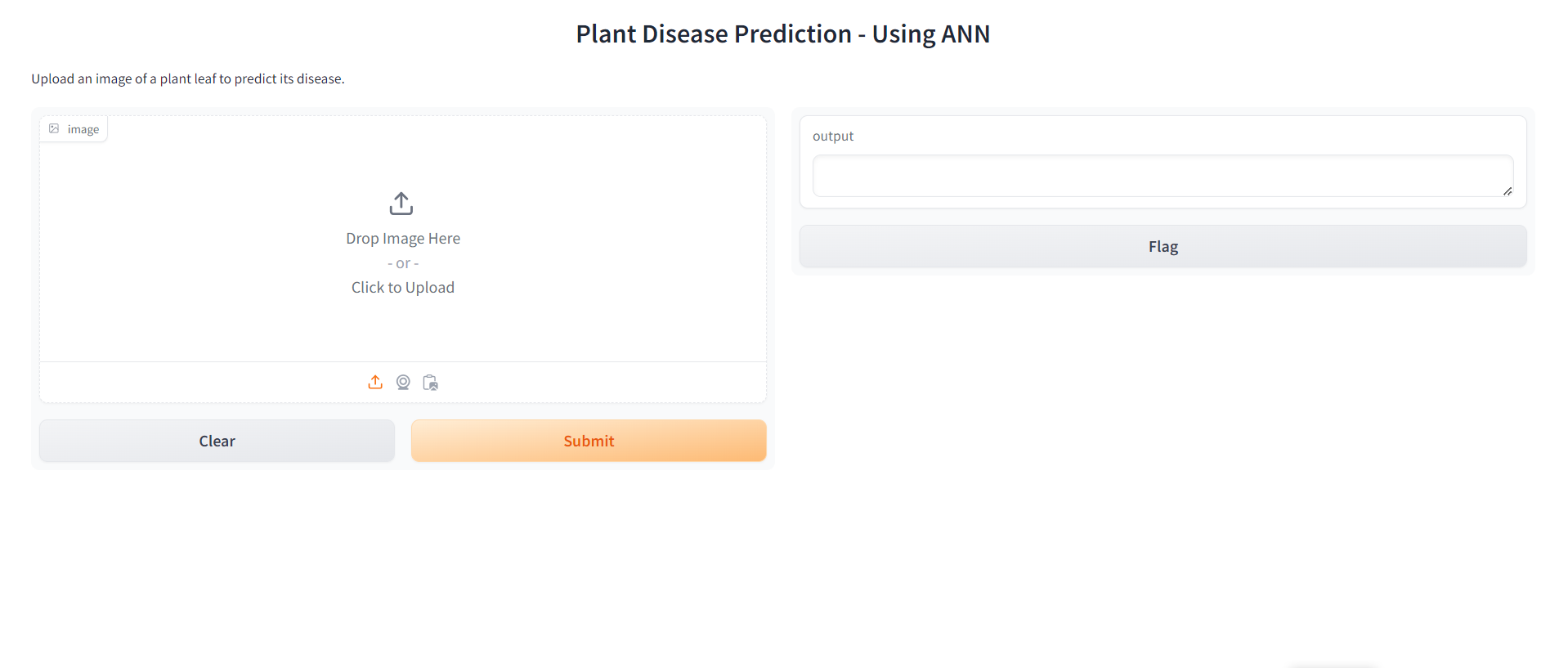


Figure (A): The Home page

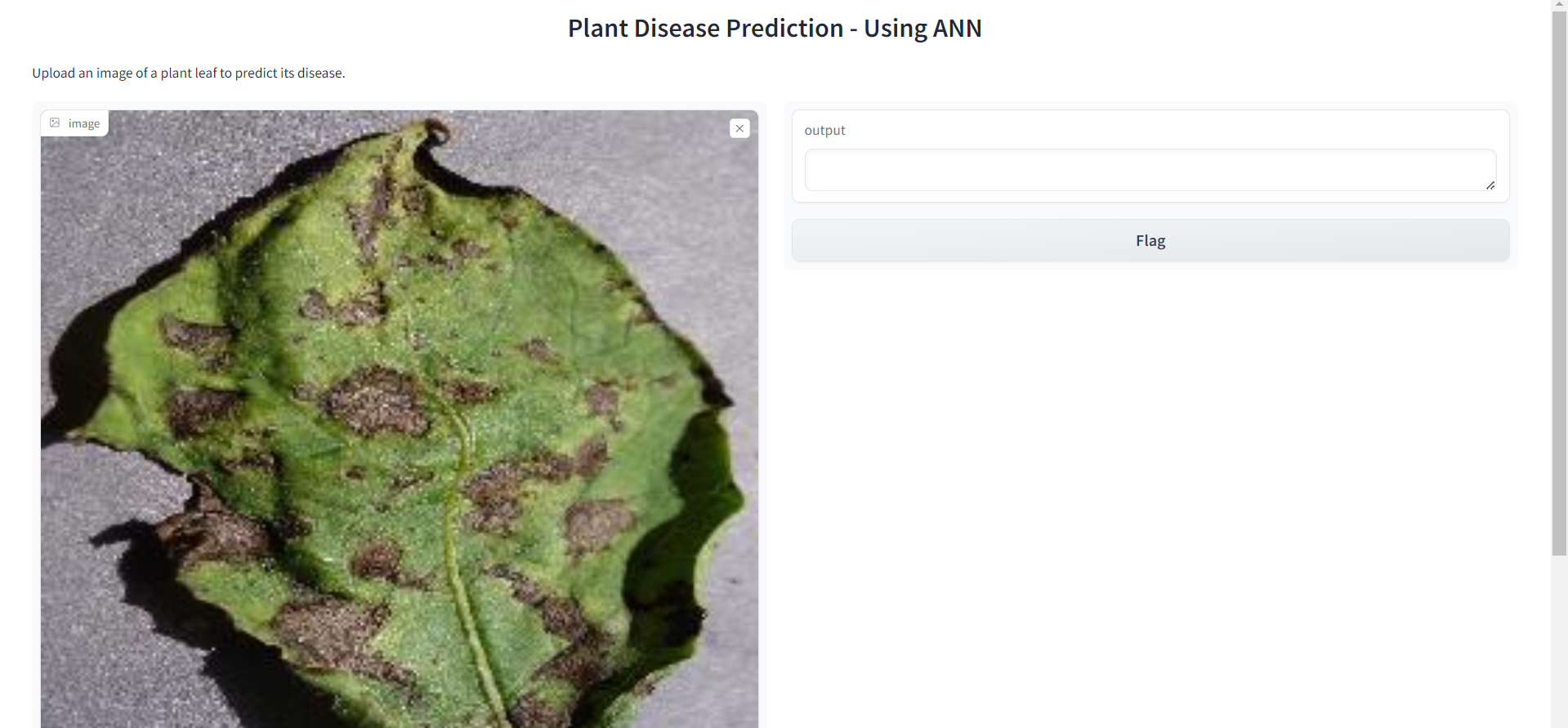
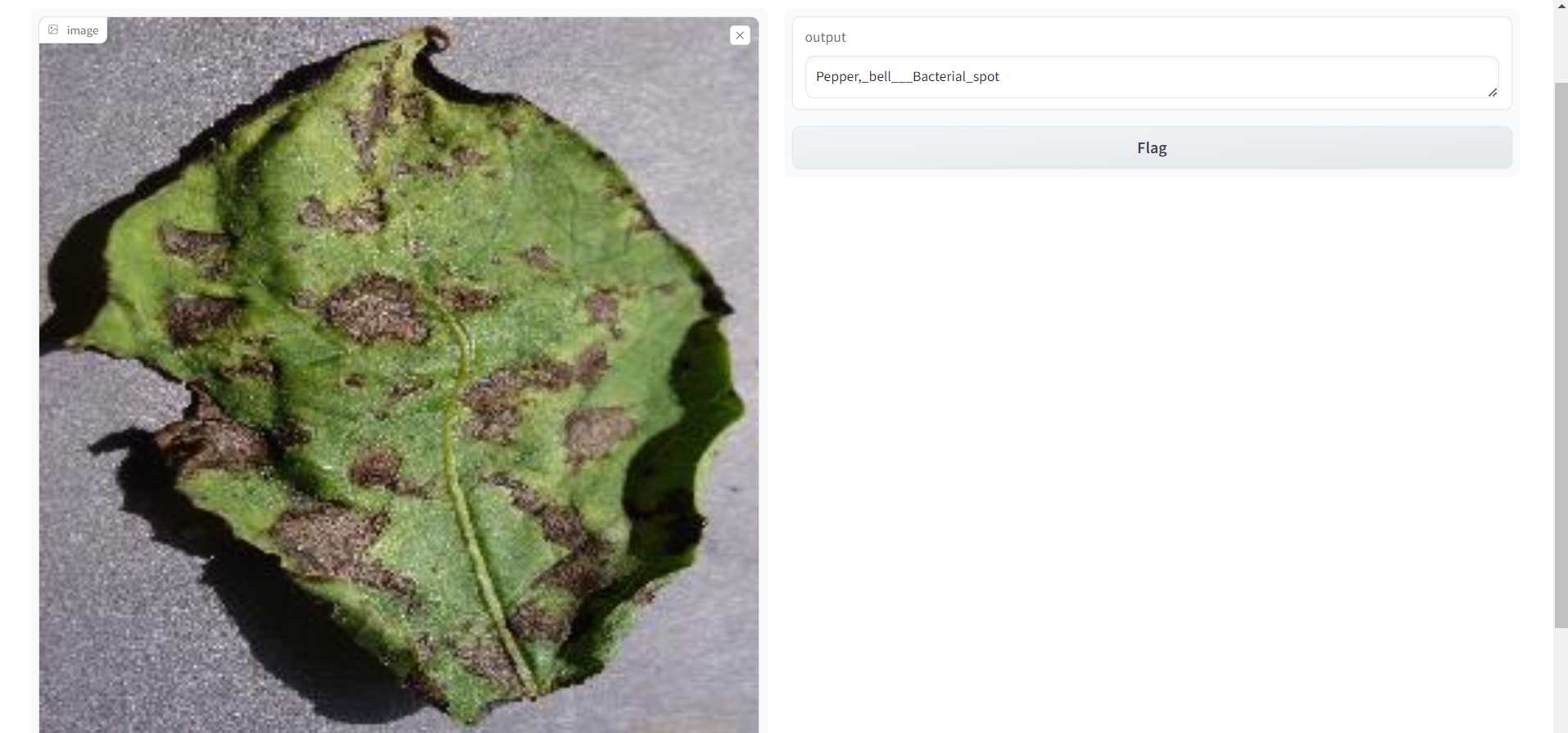


Figure (B): After uploading the image of the infected leaf



Figure(C): Predicts the disease