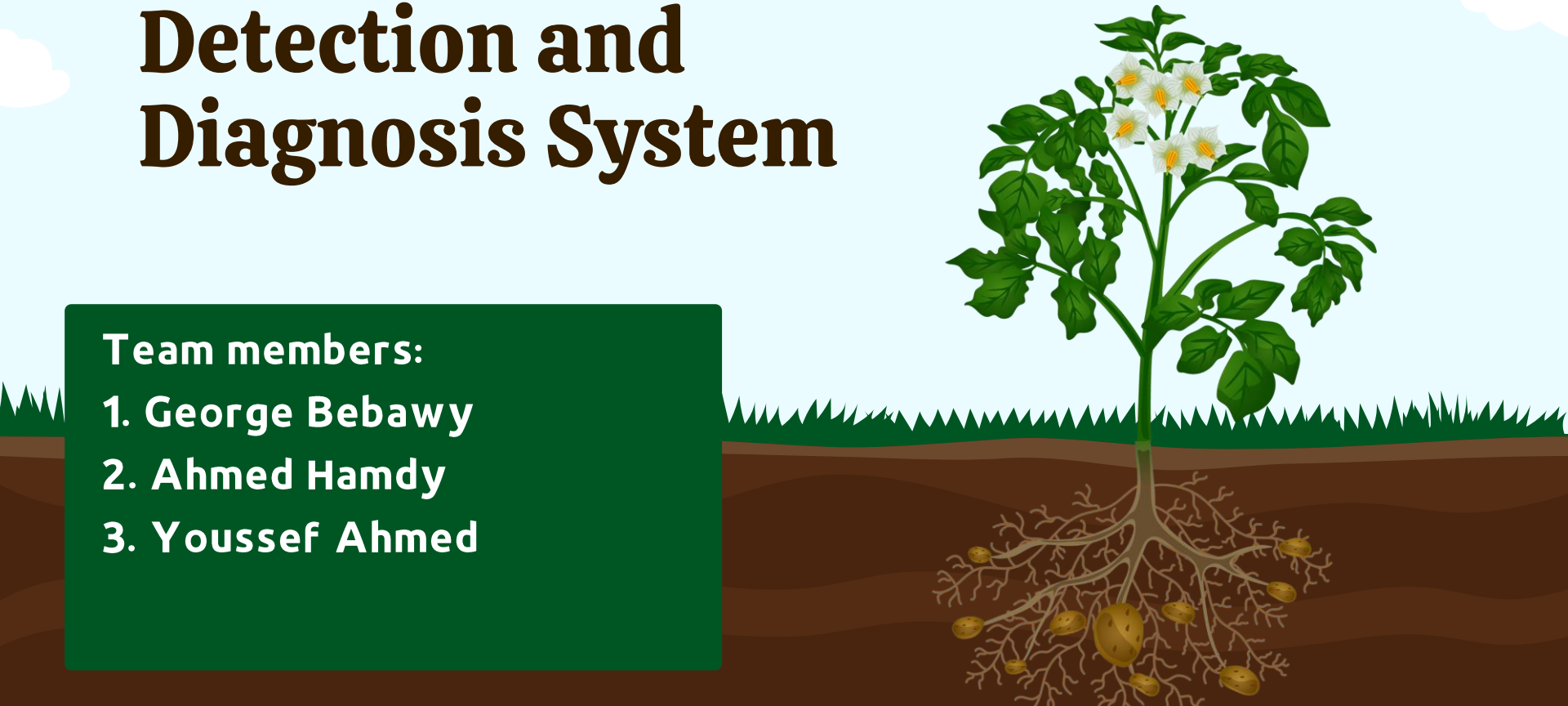
Plant Disease Detection and Diagnosis System



Team members:

1. George Bebawy
2. Ahmed Hamdy
3. Youssef Ahmed

Table of contents

01

Intro & objectives

Introduction

Our project, **Plant Disease Detection and Diagnosis System**, is designed to assist farmers, gardeners, and plant enthusiasts by leveraging AI to analyze images of plants and provide crucial insights about their health. The system functions in three key steps:

1. **Identifying the Plant Type**: Using image recognition, the system determines the species of the plant and provides a confidence percentage. For instance, it might predict with 90% certainty that the plant is an apple tree.
2. **Assessing Plant Health**: Once the plant type is identified, the system analyzes whether the plant is healthy or displaying symptoms of disease.
3. **Diagnosing the Disease**: If the plant is sick, the system identifies the specific disease and provides a detailed description of it.

By making plant health analysis quick and accessible, our goal is to help protect crops, reduce plant loss, and ensure healthier harvests.

Objective

The primary objective of this project is to develop a system capable of:

1. **Identifying Plant Types**: Determine plant species from images with high confidence.
2. **Detecting Plant Health**: Quickly assess if a plant is healthy or sick.
3. **Diagnosing Plant Diseases**: Identify specific diseases and provide detailed information for timely treatment.
4. **Displaying Confidence Levels**: Offer percentage-based confidence in predictions to enhance accuracy and decision-making.

This solution aims to provide valuable insights, improve agricultural productivity, and contribute to sustainable farming practices.

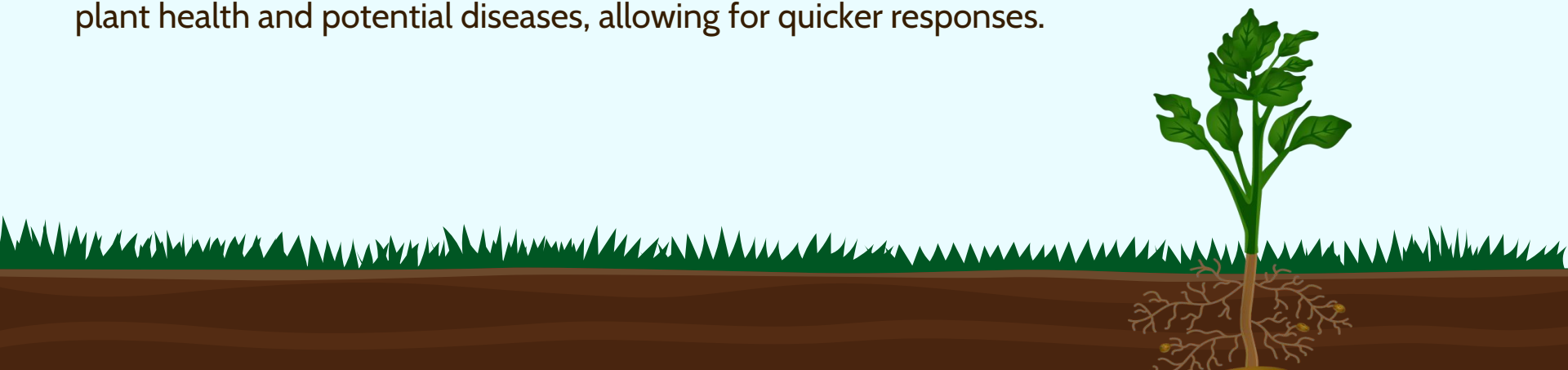
02

Goals of the Project

Goals

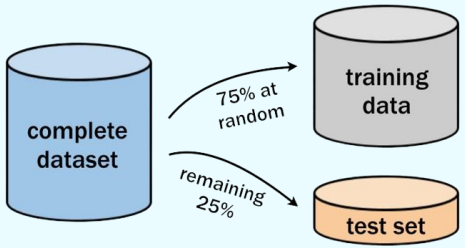
User-Friendly Experience:
Ensure that the platform is easy to use, providing farmers, gardeners, and plant lovers with a simple yet powerful tool for managing plant health.

Facilitate Real-Time Disease Detection:
Enable the system to provide real-time analysis so that users can receive instant feedback on plant health and potential diseases, allowing for quicker responses.



03

Dataset Overview



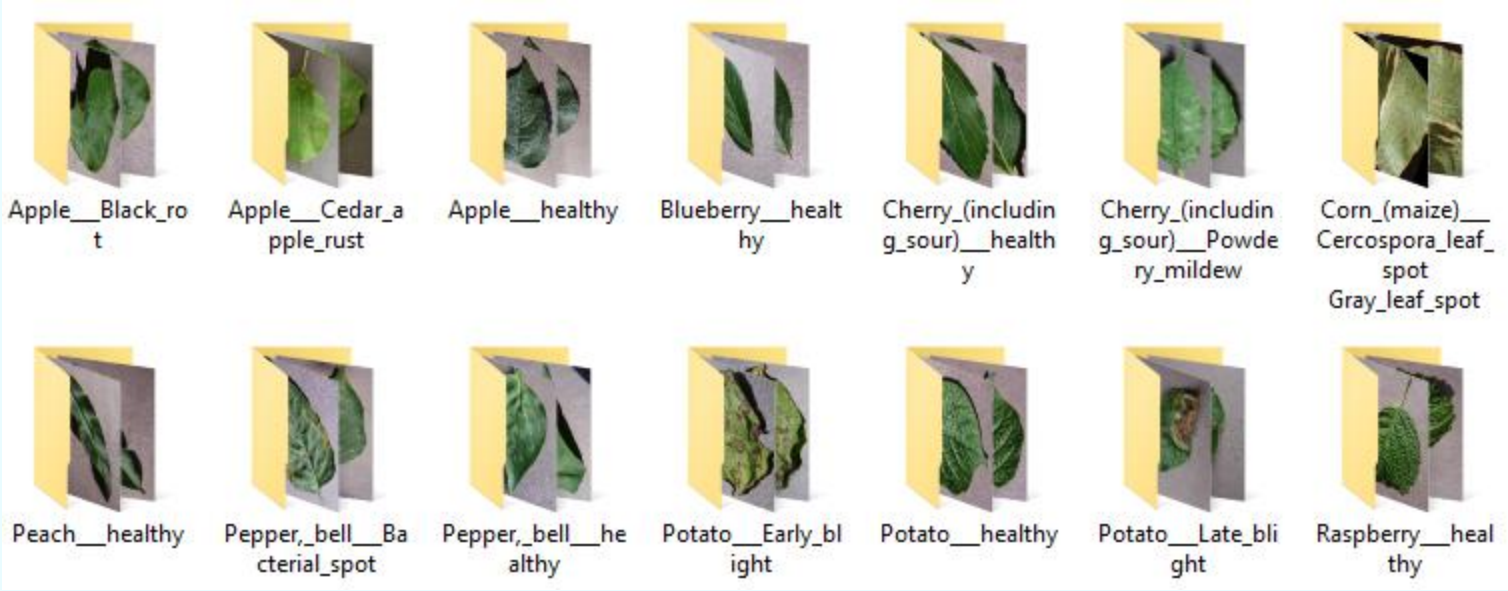
Dataset Overview

For the **Plant Disease Detection and Diagnosis System**, we have utilized the New Plant Diseases Dataset from Kaggle. This dataset plays a crucial role in training and testing the model's ability to accurately identify plant types and diagnose diseases.

Key features of the dataset include:

- **Total Images**: The dataset consists of over **87,000 images** of healthy and diseased plant leaves.
- **Plant Categories**: It covers **38 distinct classes**, representing both healthy and diseased plants from various species such as apple, cherry, corn, and grape.
- **Disease Types**: The dataset includes images of plants affected by various diseases like **Apple Scab, Black Rot, and Late Blight**, among others.
- **High-Quality Images**: All images are in **RGB format** and are of high resolution, which ensures that the model can extract fine details for better classification.
- **Diverse Conditions**: The dataset includes images captured in different conditions, adding variability in lighting, angles, and environmental factors, which helps the model generalize well across real-world scenarios.

Dataset Overview



Dataset Diseases

Apple

Apple-scab

Black-rot

Cedar-apple-rust

healthy

Corn

Common-rust

Northern-Leaf-Blight

Gray-leaf-spot

healthy

Tomato

Leaf-Mold

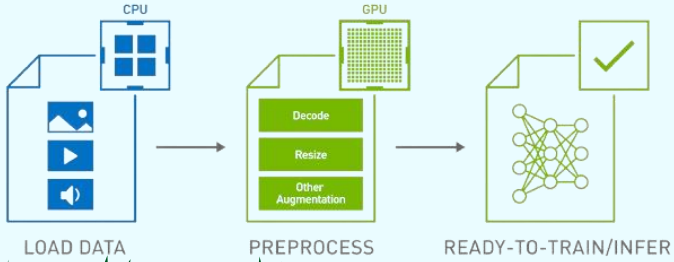
Septoria-leaf-spot

Target-Spot

healthy

04

Data Preprocessing



Data Preprocessing

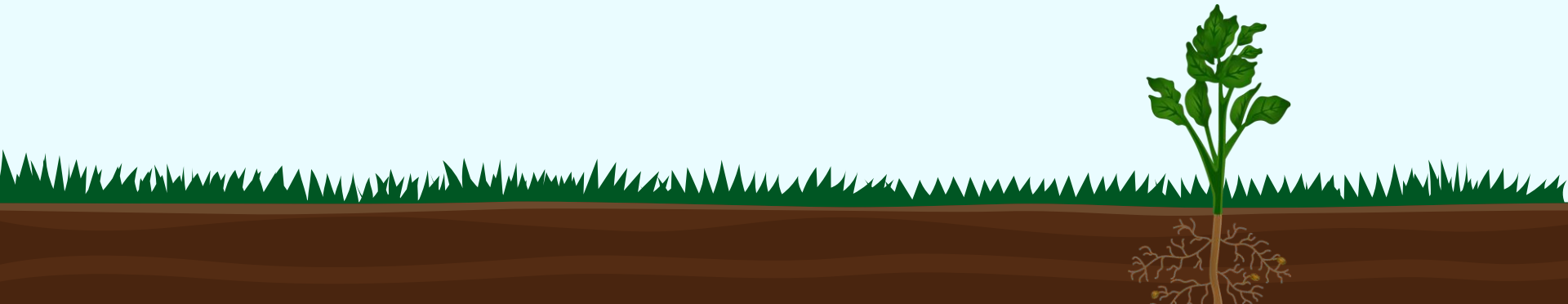
Data Preprocessing

Effective data preprocessing is a critical step in ensuring that the model can accurately classify plant types and detect diseases.

Dataset Splitting

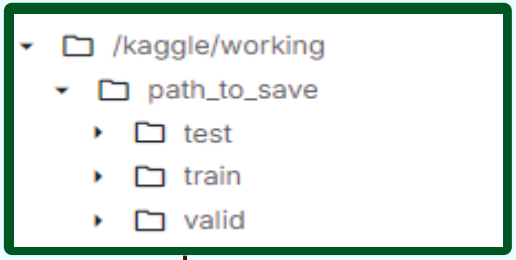
The entire dataset is divided into three main subsets to ensure the model is trained, validated, and tested effectively

Training -- Valid -- Test



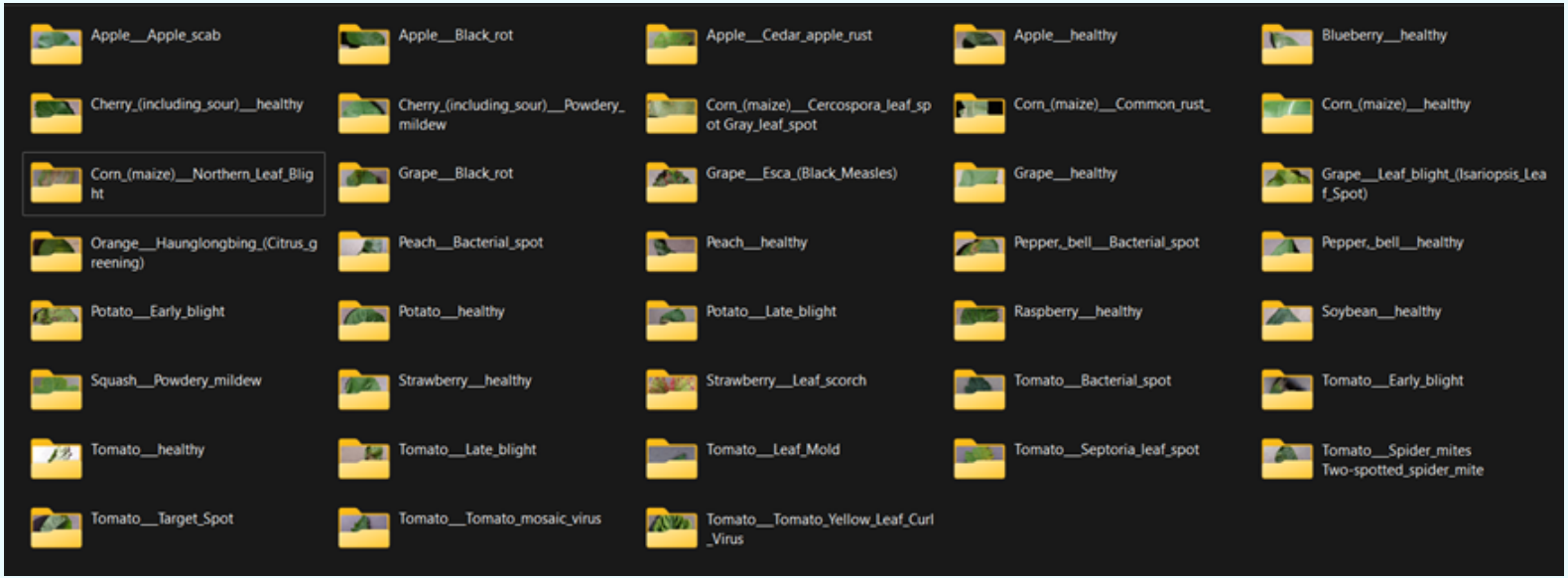
Dataset Splitting

```python
import os
import shutil
from sklearn.model_selection import train_test_split

base_dir = '/kaggle/input/plantvillage-dataset/color'
train_dir = 'path_to_save/train'
valid_dir = 'path_to_save/valid'
test_dir = 'path_to_save/test'

os.makedirs(train_dir, exist_ok=True)
os.makedirs(valid_dir, exist_ok=True)
os.makedirs(test_dir, exist_ok=True)

classes = os.listdir(base_dir)

test_size_ratio = 33 / len(classes)
train_valid_classes, test_classes = train_test_split(classes, test_size=test_size_ratio, random_state=42)

valid_size_ratio = 0.2
train_classes, valid_classes = train_test_split(train_valid_classes, test_size=valid_size_ratio, random_state=42)

def copy_files(class_list, source_dir, dest_dir):
    for class_name in class_list:
        class_dir = os.path.join(source_dir, class_name)
        dest_class_dir = os.path.join(dest_dir, class_name)
        shutil.copytree(class_dir, dest_class_dir)

copy_files(train_classes, base_dir, train_dir)
copy_files(valid_classes, base_dir, valid_dir)
copy_files(test_classes, base_dir, test_dir)
```



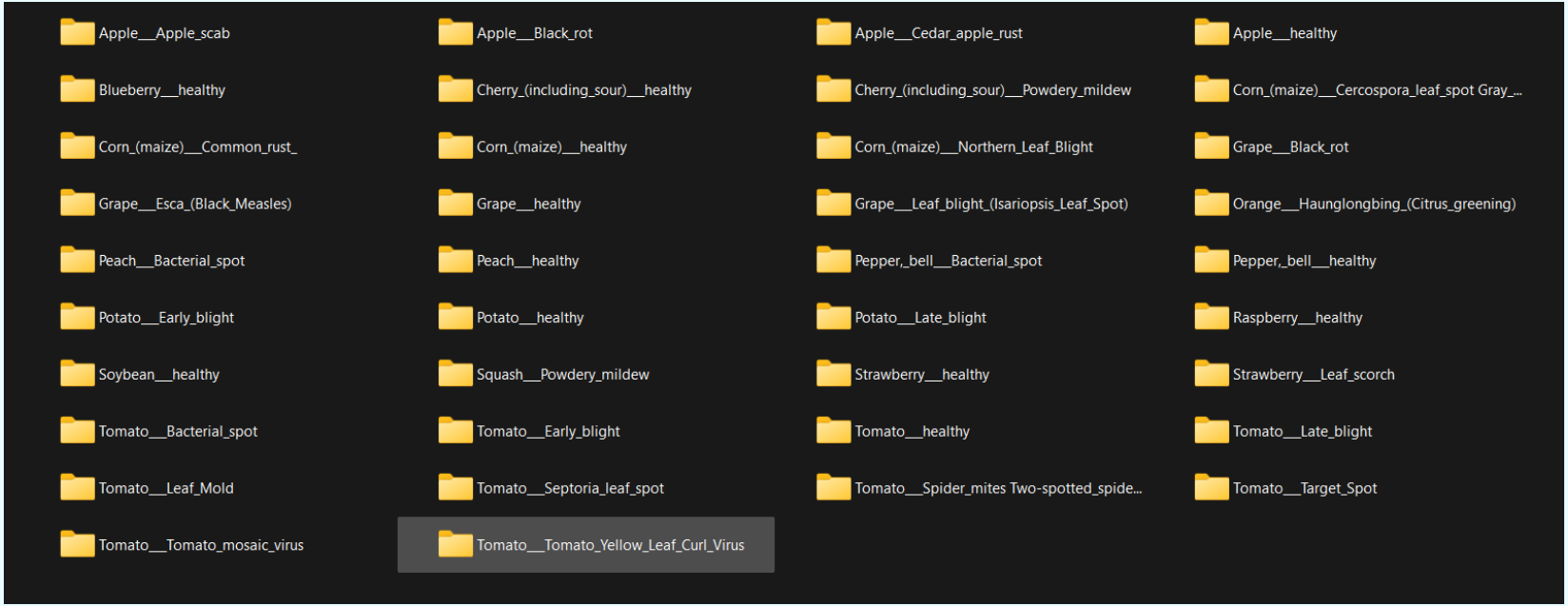
Dataset Overview

38 different classes “Training data set”



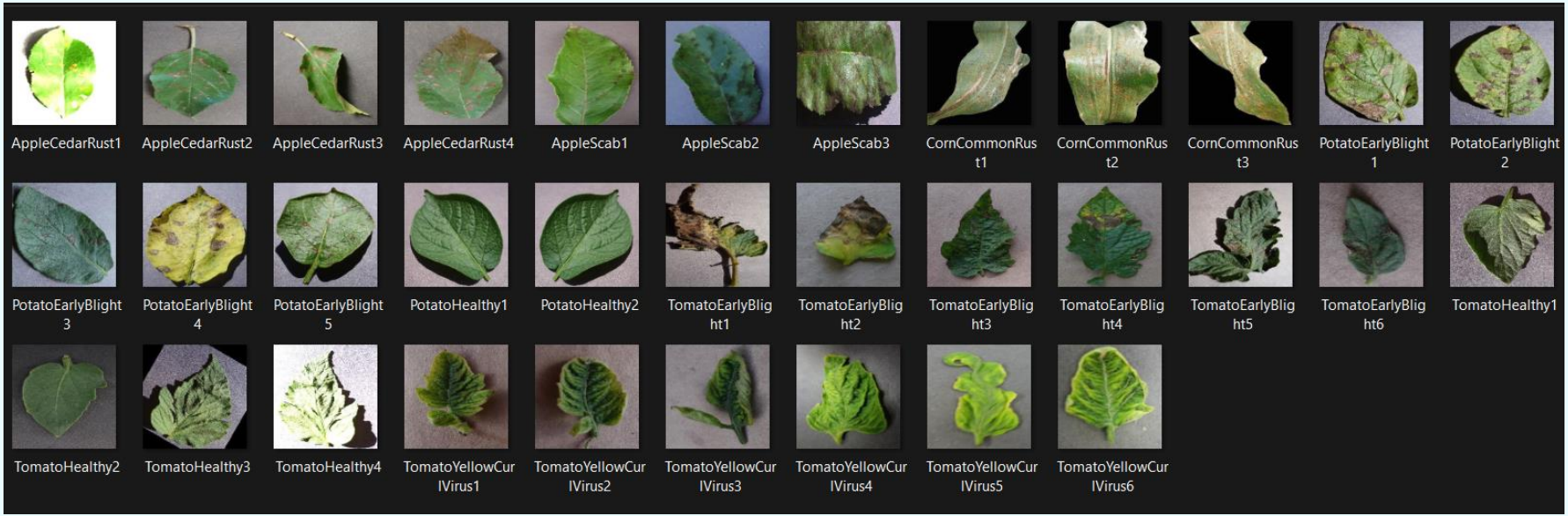
Dataset Overview

38 different classes “Validation data set”



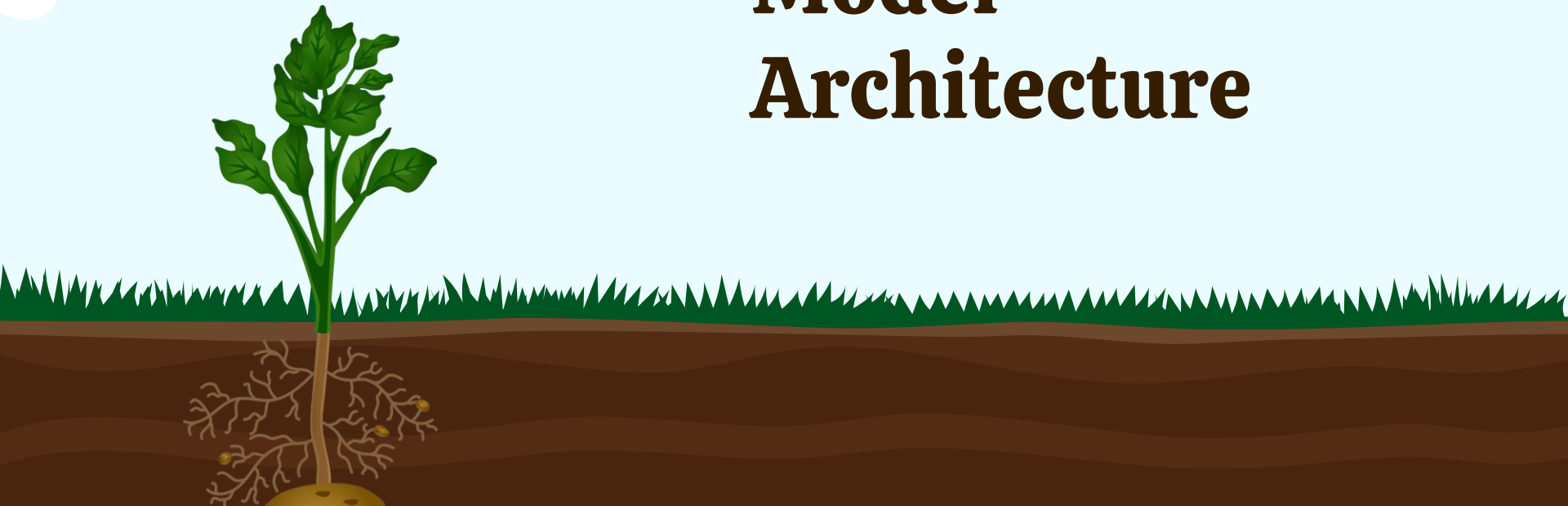
Dataset Overview

Testing Images



05

Model Architecture



Model Architecture

Import library

```
import tensorflow as tf
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import tensorflow as tf
from tensorflow.keras import layers, models
```

Data Preprocessing

```
# first We Load training dataset #
```

```
training_set = tf.keras.utils.image_dataset_from_directory(
    'train',
    labels="inferred",
    label_mode="categorical",
    class_names=None,
    color_mode="rgb",
    batch_size=32,
    image_size=(128, 128),
    shuffle=True,
    seed=None,
    validation_split=None,
    subset=None,
    interpolation="bilinear",
    follow_links=False,
    crop_to_aspect_ratio=False
)
print(f'Found {training_set.cardinality().numpy()} batches of training data')
```

```
Found 70295 files belonging to 38 classes.
Found 2197 batches of training data
```

Model Architecture

```
# secound We Load validation dataset #
```

```
validation_set = tf.keras.utils.image_dataset_from_directory(
    'valid',
    labels="inferred",
    label_mode="categorical",
    class_names=None,
    color_mode="rgb",
    batch_size=32,
    image_size=(128, 128),
    shuffle=True,
    seed=None,
    validation_split=None,
    subset=None,
    interpolation="bilinear",
    follow_links=False,
    crop_to_aspect_ratio=False
)
print(f'Found {validation_set.cardinality().numpy()} batches of validation data')
```

```
Found 17572 files belonging to 38 classes.
Found 550 batches of validation data
```

Attention Mechanism

```
# Squeeze-and-Excitation block definition #
```

```
# Squeeze-and-Excitation block definition
def se_block(input_tensor, ratio=16):
    filters = input_tensor.shape[-1]  # Number of filters in the input tensor
    se_shape = (1, 1, filters)

    se = layers.GlobalAveragePooling2D()(input_tensor)
    se = layers.Reshape(se_shape)(se)
    se = layers.Dense(filters // ratio, activation='relu', use_bias=False)(se)
    se = layers.Dense(filters, activation='sigmoid', use_bias=False)(se)
    x = layers.multiply([input_tensor, se])
    return x
```

Model Architecture

CNN

CNN Architecture

```
# Input layer
inputs = layers.Input(shape=(128, 128, 3))

# First Convolutional Layer Block
x = layers.Conv2D(filters=32, kernel_size=3, padding='same', activation='relu')(inputs)
x = layers.Conv2D(filters=32, kernel_size=3, activation='relu')(x)
x = layers.MaxPooling2D(pool_size=2, strides=2)(x)

# Attention block after the first convolutional layer
x = se_block(x)
```

```
# Second Convolutional Layer Block
x = layers.Conv2D(filters=64, kernel_size=3, padding='same', activation='relu')(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation='relu')(x)
x = layers.MaxPooling2D(pool_size=2, strides=2)(x)

# Attention block after the second convolutional layer
x = se_block(x)
```

```
# Third Convolutional Layer Block
x = layers.Conv2D(filters=128, kernel_size=3, padding='same', activation='relu')(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation='relu')(x)
x = layers.MaxPooling2D(pool_size=2, strides=2)(x)
```

```
# Fourth Convolutional Layer Block
x = layers.Conv2D(filters=256, kernel_size=3, padding='same', activation='relu')(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation='relu')(x)
x = layers.MaxPooling2D(pool_size=2, strides=2)(x)
```

```
# Fifth Convolutional Layer Block
x = layers.Conv2D(filters=512, kernel_size=3, padding='same', activation='relu')(x)
x = layers.Conv2D(filters=512, kernel_size=3, activation='relu')(x)
x = layers.MaxPooling2D(pool_size=2, strides=2)(x)
```

Model Architecture

DNN

DNN Block

```
# Flattening
x = layers.Dropout(0.25)(x)
x = layers.Flatten()(x)

# DNN Block (Dense Layer)
x = layers.Dense(units=1500, activation='relu')(x)
x = layers.Dropout(0.4)(x)
```

```
# Output Layer
outputs = layers.Dense(units=38, activation='softmax')(x)
```

Model Compilation and Summary

```
# Compiling the model with the current Adam optimizer
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Model summary
model.summary()
```

Model Summary

Layer (type)	Output Shape	Param #	Connected to
input_layer_1 (InputLayer)	(None, 128, 128, 3)	0	-
conv2d_2 (Conv2D)	(None, 128, 128, 32)	896	input_layer_1[0]…
conv2d_3 (Conv2D)	(None, 126, 126, 32)	9,248	conv2d_2[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 63, 63, 32)	0	conv2d_3[0][0]
global_average_poo… (GlobalAveragePool…	(None, 32)	0	max_pooling2d_1[…
reshape_1 (Reshape)	(None, 1, 1, 32)	0	global_average_p…
dense_2 (Dense)	(None, 1, 1, 2)	64	reshape_1[0][0]
dense_3 (Dense)	(None, 1, 1, 32)	64	dense_2[0][0]
multiply_1 (Multiply)	(None, 63, 63, 32)	0	max_pooling2d_1[… dense_3[0][0]
conv2d_4 (Conv2D)	(None, 63, 63, 64)	18,496	multiply_1[0][0]
conv2d_5 (Conv2D)	(None, 61, 61, 64)	36,928	conv2d_4[0][0]
max_pooling2d_2 (MaxPooling2D)	(None, 30, 30, 64)	0	conv2d_5[0][0]
global_average_poo… (GlobalAveragePool…	(None, 64)	0	max_pooling2d_2[…
reshape_2 (Reshape)	(None, 1, 1, 64)	0	global_average_p…
dense_4 (Dense)	(None, 1, 1, 4)	256	reshape_2[0][0]
dense_5 (Dense)	(None, 1, 1, 64)	256	dense_4[0][0]
multiply_2 (Multiply)	(None, 30, 30, 64)	0	max_pooling2d_2[… dense_5[0][0]
conv2d_6 (Conv2D)	(None, 30, 30, 128)	73,856	multiply_2[0][0]
conv2d_7 (Conv2D)	(None, 28, 28, 128)	147,584	conv2d_6[0][0]
max_pooling2d_3 (MaxPooling2D)	(None, 14, 14, 128)	0	conv2d_7[0][0]
conv2d_8 (Conv2D)	(None, 14, 14, 256)	295,168	max_pooling2d_3[…
conv2d_9 (Conv2D)	(None, 12, 12, 256)	590,080	conv2d_8[0][0]
max_pooling2d_4 (MaxPooling2D)	(None, 6, 6, 256)	0	conv2d_9[0][0]
conv2d_10 (Conv2D)	(None, 6, 6, 512)	1,180,160	max_pooling2d_4[…
conv2d_11 (Conv2D)	(None, 4, 4, 512)	2,359,808	conv2d_10[0][0]
max_pooling2d_5 (MaxPooling2D)	(None, 2, 2, 512)	0	conv2d_11[0][0]
dropout (Dropout)	(None, 2, 2, 512)	0	max_pooling2d_5[…
flatten (Flatten)	(None, 2048)	0	dropout[0][0]
dense_6 (Dense)	(None, 1500)	3,073,500	flatten[0][0]
dropout_1 (Dropout)	(None, 1500)	0	dense_6[0][0]
dense_7 (Dense)	(None, 38)	57,038	dropout_1[0][0]

Total params: 7,843,402 (29.92 MB)

Trainable params: 7,843,402 (29.92 MB)

Non-trainable params: 0 (0.00 B)

Model Summary

```
# Training the model
training_history = model.fit(x=training_set,
                             validation_data=validation_set,
                             epochs=10)
```

```
Epoch 1/10
2197/2197 ━━━━━━━━━━━━━━━━━━━━ 960s 436ms/step - accuracy: 0.3804 - loss: 2.1861 - val_accuracy: 0.8352 - val_loss: 0.5128
Epoch 2/10
2197/2197 ━━━━━━━━━━━━━━━━━━━━ 929s 423ms/step - accuracy: 0.8321 - loss: 0.5318 - val_accuracy: 0.9066 - val_loss: 0.2836
Epoch 3/10
2197/2197 ━━━━━━━━━━━━━━━━━━━━ 929s 423ms/step - accuracy: 0.9004 - loss: 0.3047 - val_accuracy: 0.9190 - val_loss: 0.2482
Epoch 4/10
2197/2197 ━━━━━━━━━━━━━━━━━━━━ 940s 428ms/step - accuracy: 0.9306 - loss: 0.2078 - val_accuracy: 0.9495 - val_loss: 0.1617
Epoch 5/10
2197/2197 ━━━━━━━━━━━━━━━━━━━━ 936s 426ms/step - accuracy: 0.9491 - loss: 0.1535 - val_accuracy: 0.9544 - val_loss: 0.1416
Epoch 6/10
2197/2197 ━━━━━━━━━━━━━━━━━━━━ 933s 425ms/step - accuracy: 0.9621 - loss: 0.1162 - val_accuracy: 0.9556 - val_loss: 0.1332
Epoch 7/10
2197/2197 ━━━━━━━━━━━━━━━━━━━━ 928s 422ms/step - accuracy: 0.9673 - loss: 0.0983 - val_accuracy: 0.9374 - val_loss: 0.2075
Epoch 8/10
2197/2197 ━━━━━━━━━━━━━━━━━━━━ 927s 422ms/step - accuracy: 0.9720 - loss: 0.0842 - val_accuracy: 0.9657 - val_loss: 0.1117
Epoch 9/10
2197/2197 ━━━━━━━━━━━━━━━━━━━━ 926s 421ms/step - accuracy: 0.9788 - loss: 0.0670 - val_accuracy: 0.9548 - val_loss: 0.1558
Epoch 10/10
2197/2197 ━━━━━━━━━━━━━━━━━━━━ 946s 431ms/step - accuracy: 0.9804 - loss: 0.0615 - val_accuracy: 0.9627 - val_loss: 0.1187
```

Cheak The accuracy

```
# Evaluate Training Set Accuracy
train_loss, train_acc = model.evaluate(training_set)
print('Training accuracy:', train_acc)

# Evaluate Validation Set Accuracy
val_loss, val_acc = model.evaluate(validation_set)
print('Validation accuracy:', val_acc)
```

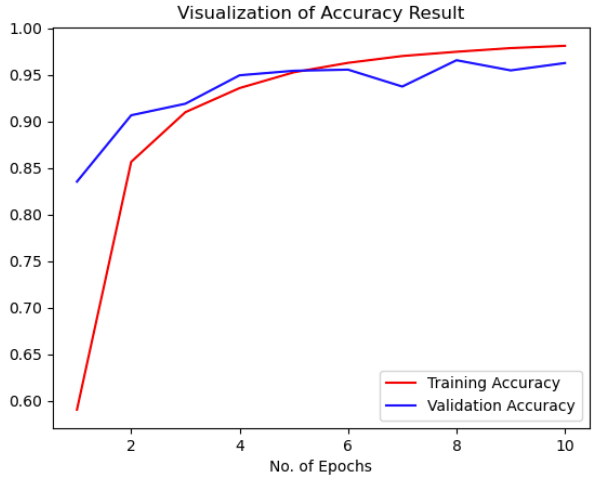
```
2197/2197 ━━━━━━━━━━━━━━━━━━━━ 228s 104ms/step - accuracy: 0.9880 - loss: 0.0335
Training accuracy: 0.9892311096191406
550/550 ━━━━━━━━━━━━━━━━━━━━ 57s 103ms/step - accuracy: 0.9629 - loss: 0.1235
Validation accuracy: 0.9626678824424744
```


```
Training accuracy: 0.9892311096191406
550/550 ━━━━━━━━━━━━━━━━━━━━ 57s 103ms/s
Validation accuracy: 0.9626678824424744
```

Model Summary

Plotting the accuracy results

```
epochs = [i for i in range(1,11)]
plt.plot(epochs,training_history.history['accuracy'],color='red',label='Training Accuracy')
plt.plot(epochs,training_history.history['val_accuracy'],color='blue',label='Validation Accuracy')
plt.xlabel('No. of Epochs')
plt.title('Visualization of Accuracy Result')
plt.legend()
plt.show()
```



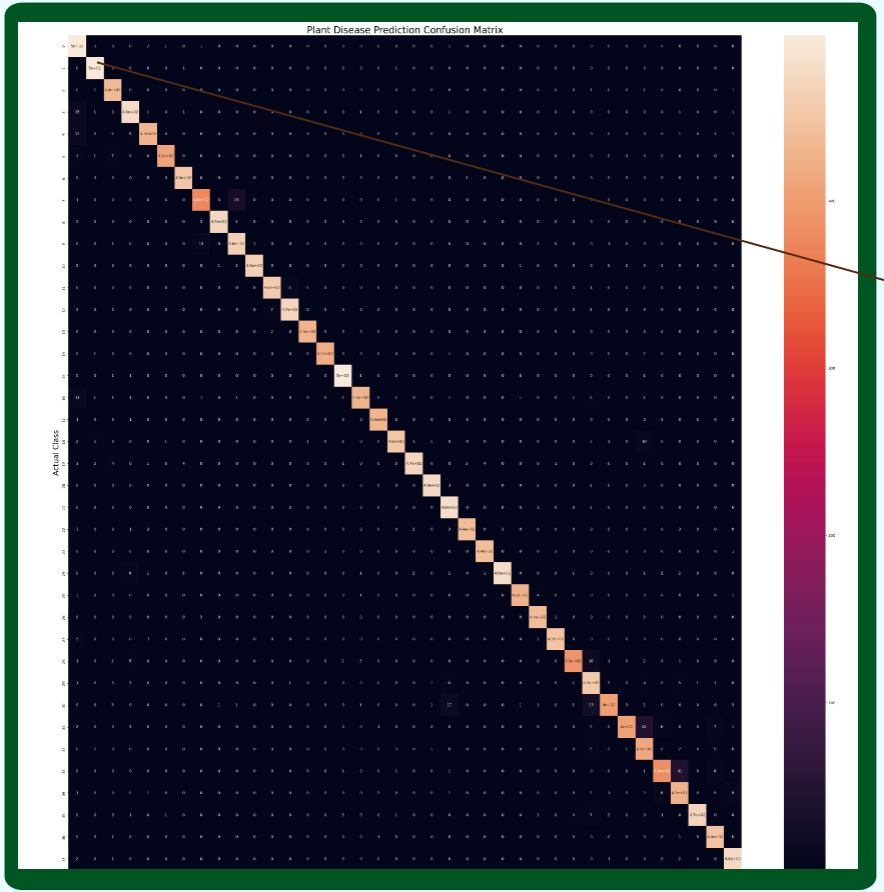
Model Summary

	precision	recall	f1-score	support
Apple___Apple_scab	0.88	0.98	0.93	504
Apple___Black_rot	0.96	1.00	0.98	497
Apple___Cedar_apple_rust	0.95	0.99	0.97	440
Apple___healthy	0.94	0.96	0.95	502
Blueberry___healthy	0.97	0.94	0.96	454
Cherry_(including_sour)___Powdery_mildew	0.99	0.97	0.98	421
Cherry_(including_sour)___healthy	0.98	0.99	0.98	456
orn_(maize)___Cercospora_leaf_spot Gray_leaf_spot	0.95	0.92	0.94	410
Corn_(maize)___Common_rust_	0.99	0.99	0.99	477
Corn_(maize)___Northern_Leaf_Blight	0.94	0.97	0.95	477
Corn_(maize)___healthy	1.00	0.99	1.00	465
Grape___Black_rot	0.98	0.97	0.98	472
Grape___Esca_(Black_Measles)	0.98	0.98	0.98	480
Grape___Leaf_blight_(Isariopsis_Leaf_Spot)	0.99	1.00	0.99	430
Grape___healthy	1.00	0.99	0.99	423
Orange___Haunglongbing_(Citrus_greening)	0.98	0.99	0.99	503
Peach___Bacterial_spot	0.98	0.94	0.96	459
Peach___healthy	0.99	1.00	0.99	432
Pepper,_bell___Bacterial_spot	0.98	0.95	0.96	478
Pepper,_bell___healthy	0.99	0.95	0.97	497
Potato___Early_blight	0.99	0.98	0.99	485
Potato___Late_blight	0.93	0.99	0.96	485
Potato___healthy	0.99	0.96	0.98	456
Raspberry___healthy	0.99	0.99	0.99	445
Soybean___healthy	0.99	0.95	0.97	505
Squash___Powdery_mildew	0.99	0.98	0.99	434
Strawberry___Leaf_scorch	0.98	0.98	0.98	444
Strawberry___healthy	0.99	0.98	0.98	456
Tomato___Bacterial_spot	0.98	0.92	0.95	425
Tomato___Early_blight	0.86	0.94	0.90	480
Tomato___Late_blight	0.97	0.87	0.92	463
Tomato___Leaf_Mold	0.99	0.86	0.92	470
Tomato___Septoria_leaf_spot	0.84	0.95	0.89	436
Tomato___Spider_mites Two-spotted_spider_mite	0.96	0.88	0.92	435
Tomato___Target_Spot	0.88	0.93	0.90	457
Tomato___Tomato_Yellow_Leaf_Curl_Virus	1.00	0.96	0.98	490
Tomato___Tomato_mosaic_virus	0.95	0.99	0.97	448
Tomato___healthy	0.97	0.99	0.98	481
accuracy			0.96	17572
macro avg	0.96	0.96	0.96	17572
weighted avg	0.96	0.96	0.96	17572

	precision	recall	f1-score	support
Apple___Apple_scab	0.88	0.98	0.93	504

Model Summary

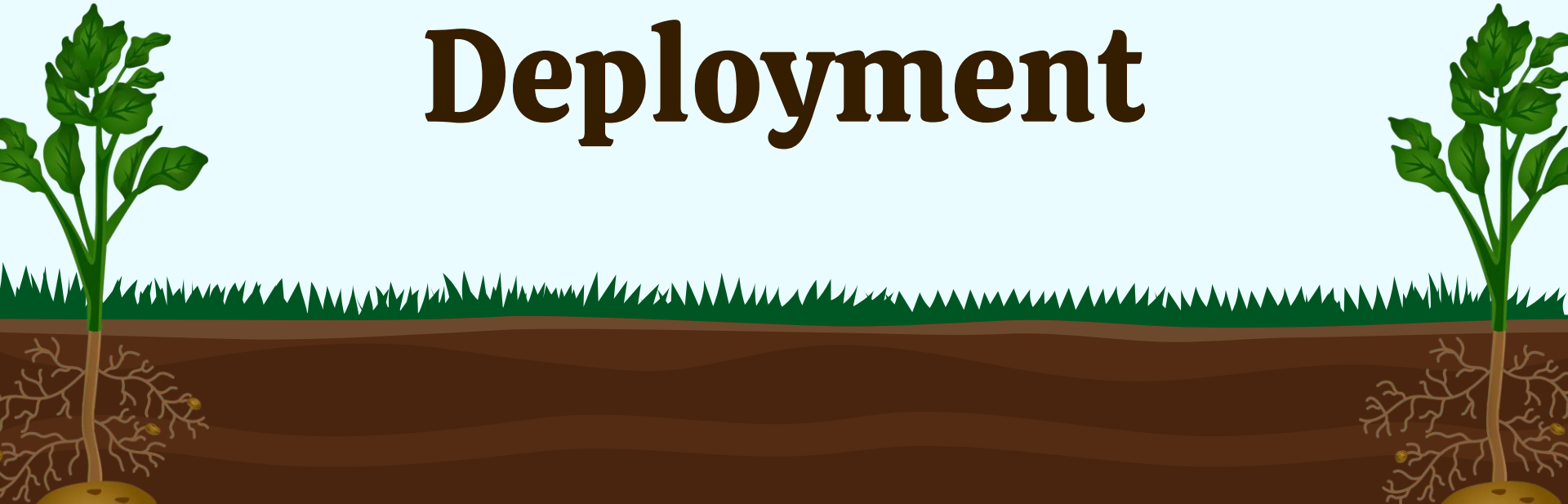
C.M



	Predicted 0	Predicted 1	Predicted 2	Predicted 3
Actual 0	500	3	0	2
Actual 1	0	497	0	0
Actual 2	0	1	440	0
Actual 3	15	1	0	459

06

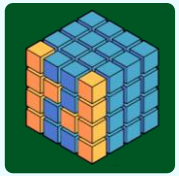
Model Deployment



Model Deployment

Streamlit

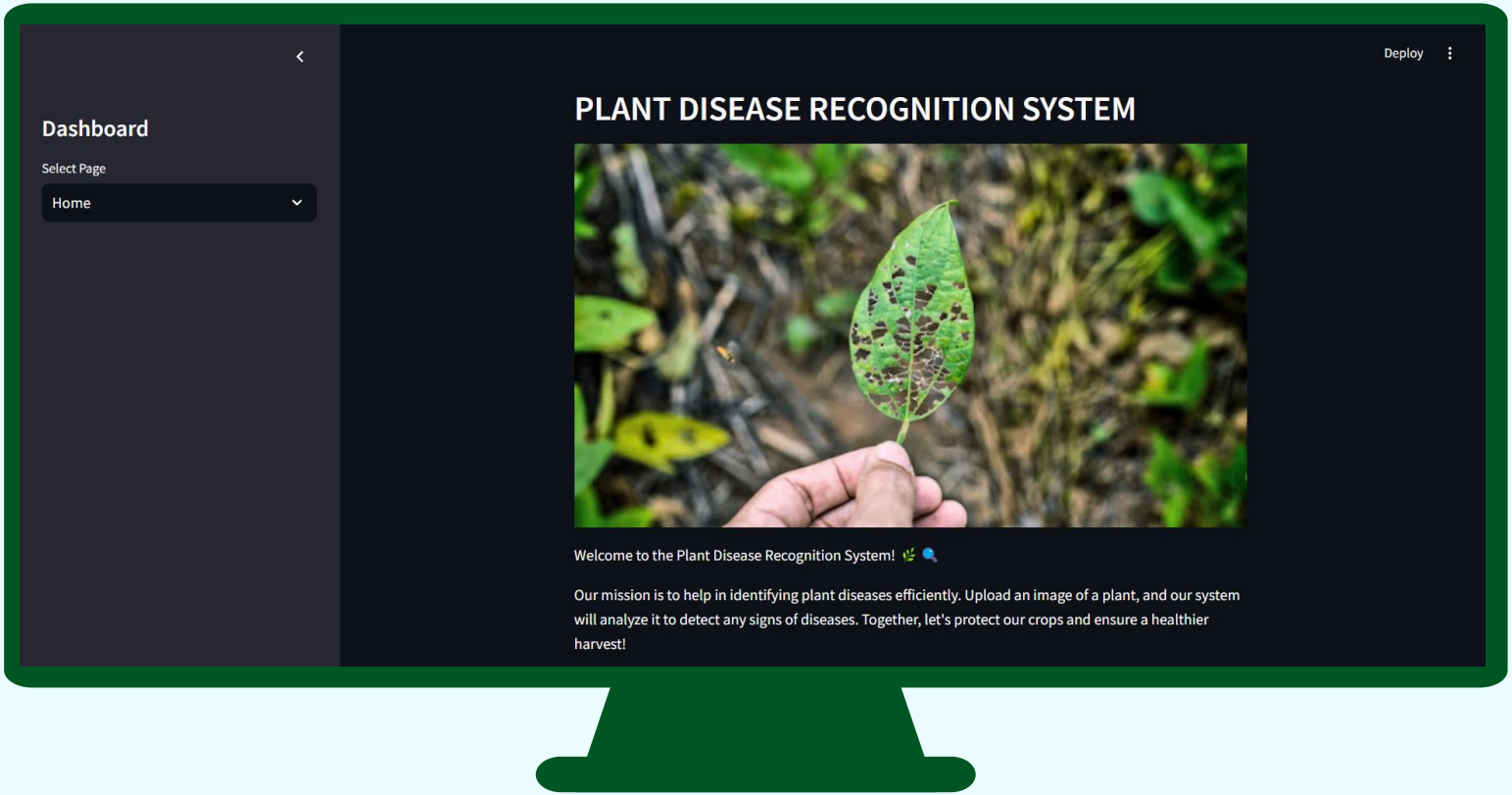
Tensor Flow



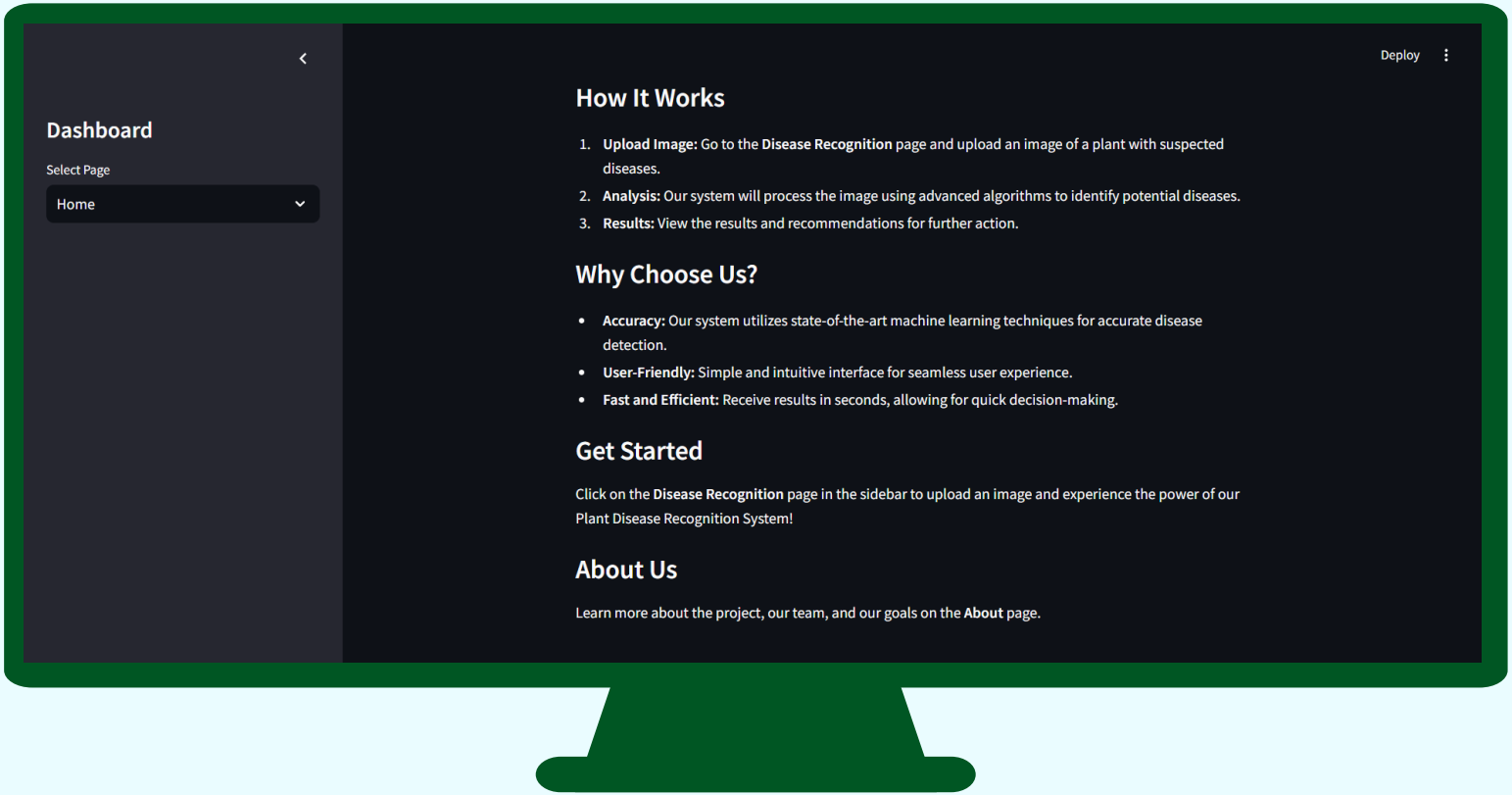
NumPy

Keras

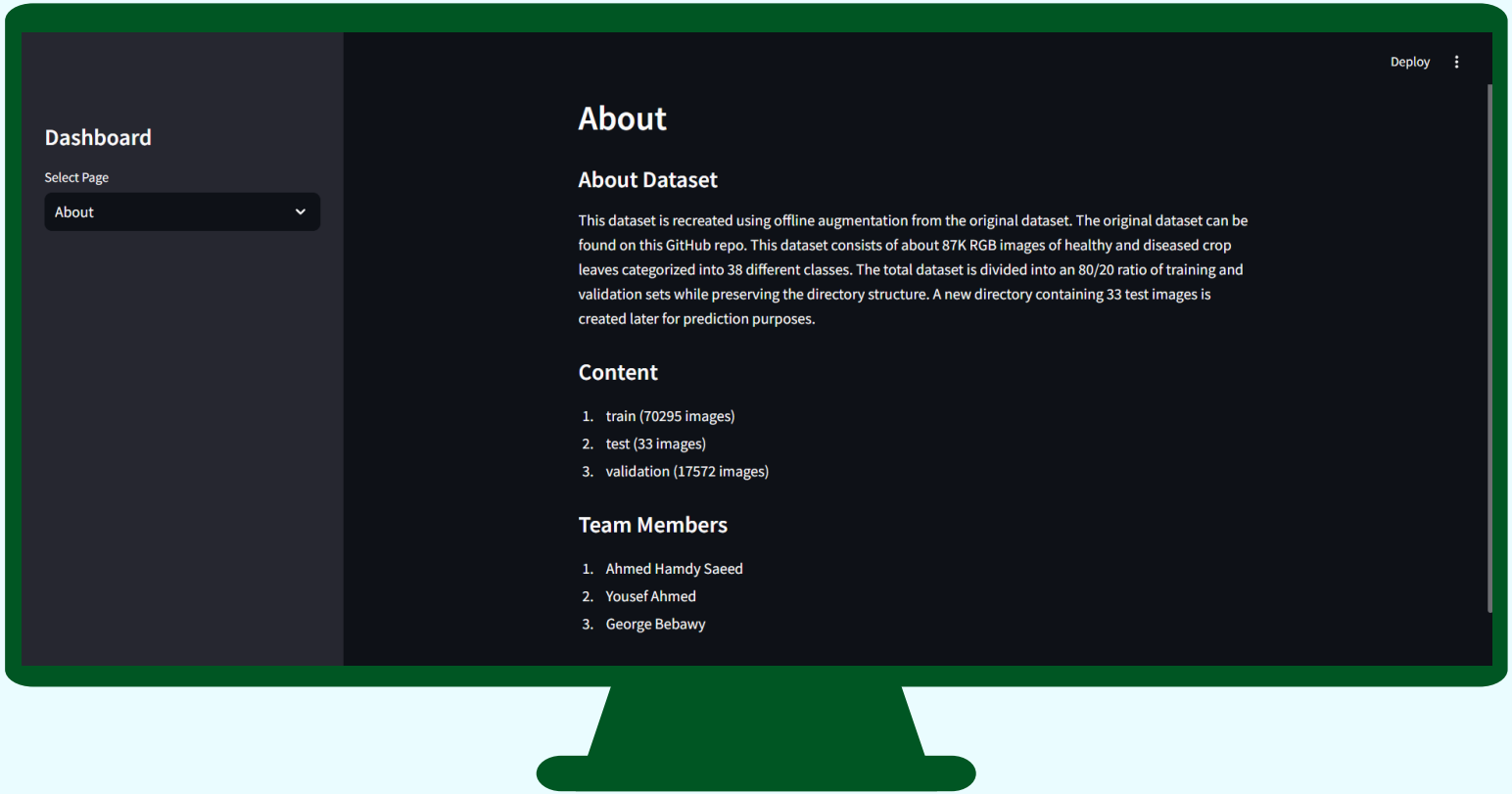
Model Deployment



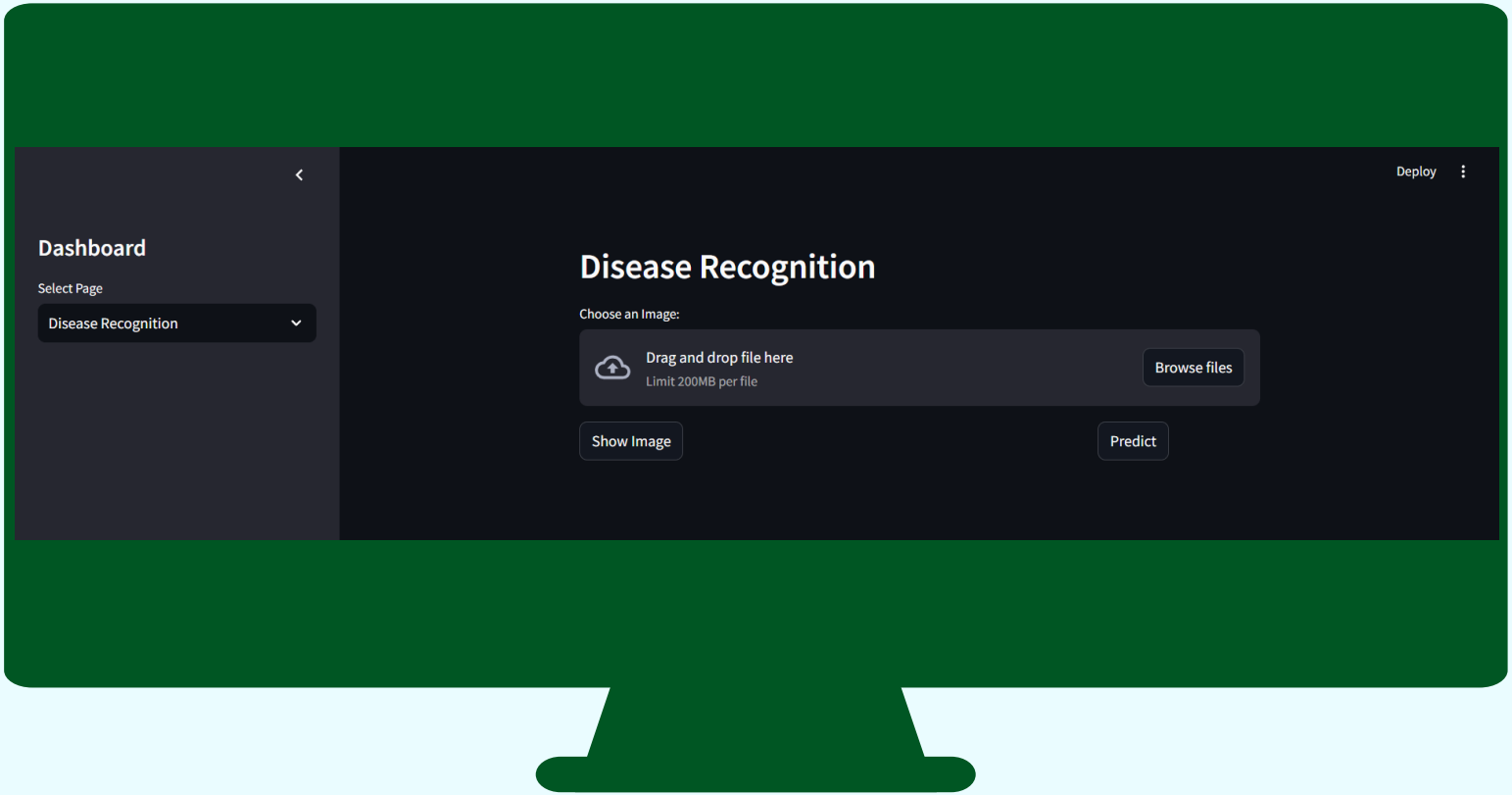
Model Deployment



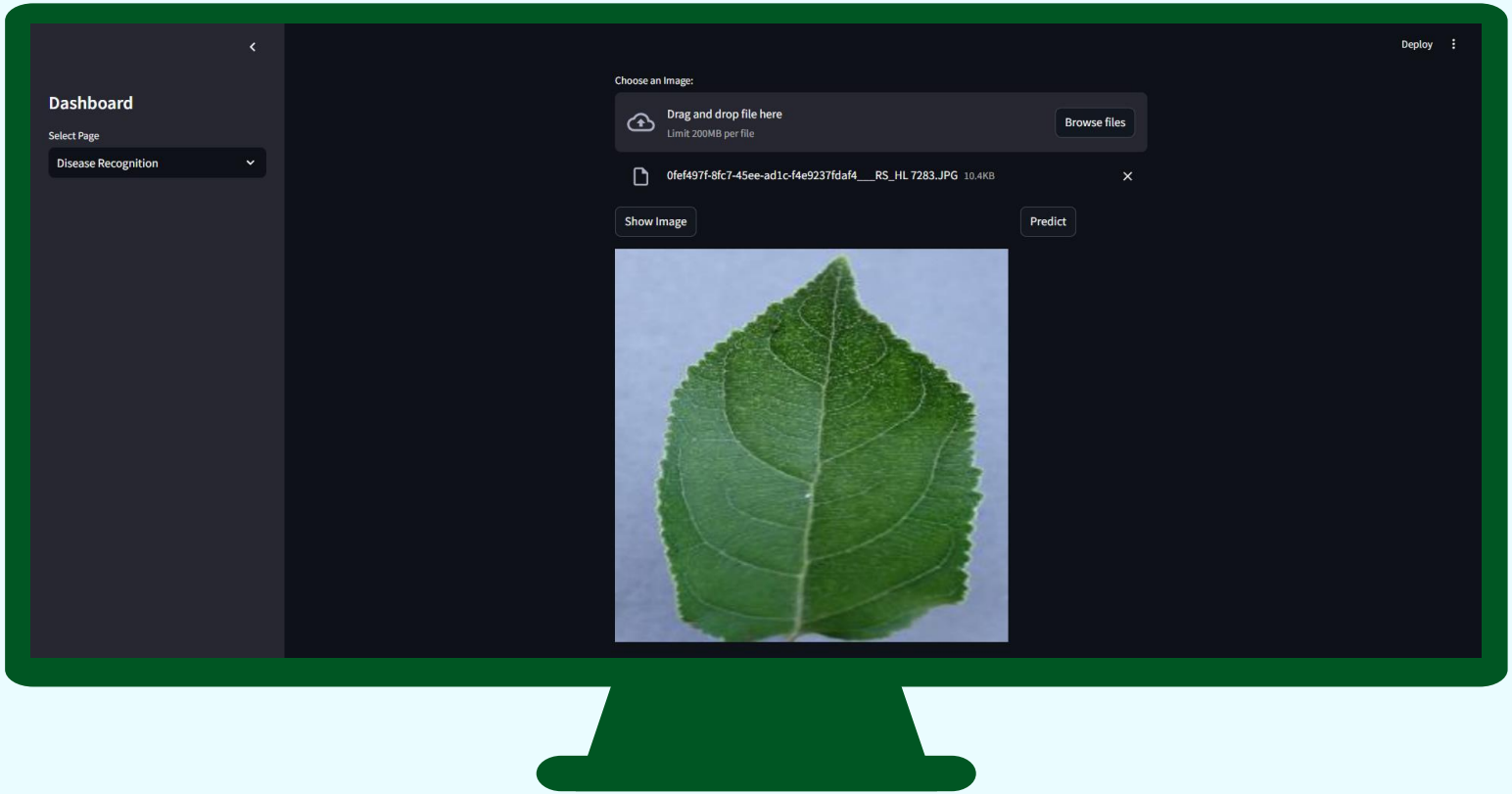
Model Deployment



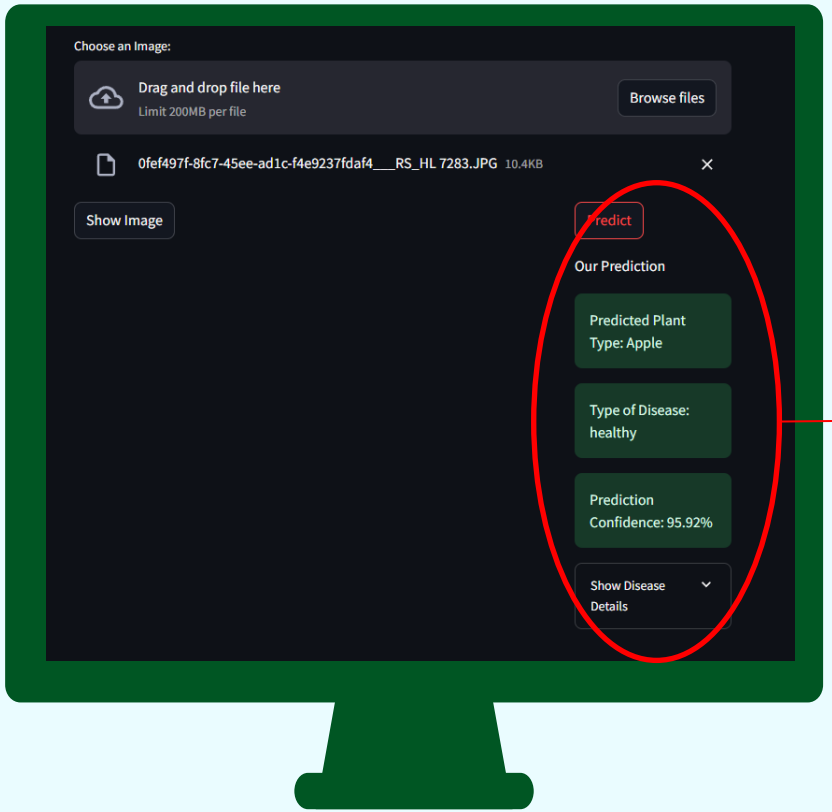
Model Deployment

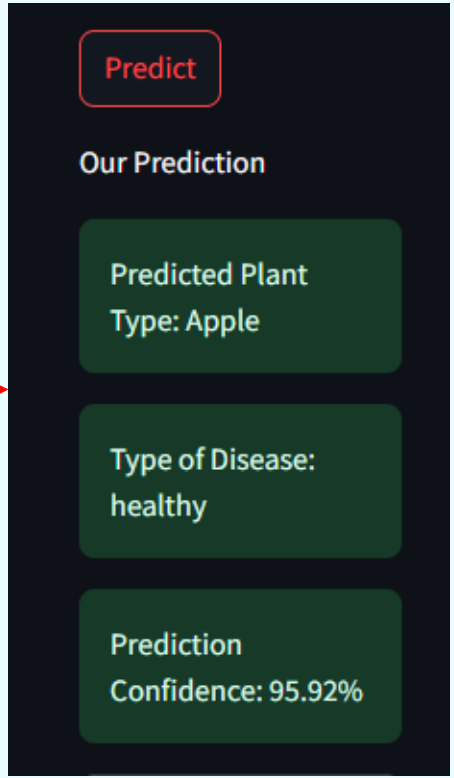


Model Deployment



Model Deployment





Thanks!

Do you have any questions?