Imports

In [ ]:

```
import os
import numpy as np
import pandas as pd
import seaborn as sns
import cv2
import random

import matplotlib.pyplot as plt
from plotly.subplots import make_subplots
import plotly.graph_objects as go
from plotly.offline import iplot
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
import itertools
import missingno as msno
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam, Adamax
from tensorflow.keras.metrics import categorical_crossentropy
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Activat
ion, Dropout, BatchNormalization
from tensorflow.keras import regularizers
from keras.callbacks import EarlyStopping, LearningRateScheduler
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.efficientnet import preprocess_input
from tensorflow.keras.applications import EfficientNetB0
from tensorflow.keras.models import load_model
```

Data Import

In [5]:

```
#Giving the Directory name for dataset
data_directory = 'D:/Big Data Analytics/Term-2/BDM 3014 - Introduction to Artific
ial Intelligence 01/Final Project/MangoLeafBD Dataset'
#Giving name to the dataset for EDAs
dataset_name = 'Mango Leaf Disease Dataset'
```

In [6]:

```
#Function to get data paths and label
def get_data_paths(data_directory):

    #Initializing lists to store data paths and labels
    filepaths = []
    labels = []

    #Getting all the folders from the given directory
    folds = os.listdir(data_directory)

    #Looping through each folder whihch represents labels
    for fold in folds:
        foldpath = os.path.join(data_directory, fold)
        filelist = os.listdir(foldpath)
        for file in filelist:
            fpath = os.path.join(foldpath, file)
            filepaths.append(fpath)
```

```
            labels.append(fold)

    return filepaths, labels


#Calling the function
filepaths, labels = get_data_paths(data_directory)
```

In [7]:

```
#function to create one dataframe with both file paths and labels
def create_df(filepaths, labels):

    Fseries = pd.Series(filepaths, name= 'filepaths')
    Lseries = pd.Series(labels, name='labels')
    df = pd.concat([Fseries, Lseries], axis= 1)

    return df

df = create_df(filepaths, labels)
```

In [8]:

```
#Printing the dataframe to check
df.head()
```

Out[8]:

	filepaths	labels
0	D:/Big Data Analytics/Term-2/BDM 3014 - Introd...	Anthracnose
1	D:/Big Data Analytics/Term-2/BDM 3014 - Introd...	Anthracnose
2	D:/Big Data Analytics/Term-2/BDM 3014 - Introd...	Anthracnose
3	D:/Big Data Analytics/Term-2/BDM 3014 - Introd...	Anthracnose
4	D:/Big Data Analytics/Term-2/BDM 3014 - Introd...	Anthracnose

In [9]:

```
#Function to check the datasize and classes inside the dataset
def num_from_dataset(df, name='df'):
    print(f"The {name} has {df.shape[0]} images.")
    print(f"The {name} has {len(df['labels'].unique())} classes")
num_from_dataset(df, dataset_name)
```

```
The Mango Leaf Disease Dataset has 4000 images.
The Mango Leaf Disease Dataset has 8 classes
```

In [10]:

```
#Function to count images inside particular labels
def classes_count(df, name='df'):

    print(f"The {name} has: ")

    print()
    for name in df['labels'].unique():
        num_class = len(df['labels'][df['labels'] == name])
        print(f"'{name}' has {num_class} images")

classes_count(df, dataset_name)
```

```
The Mango Leaf Disease Dataset has:

'Anthracnose' has 500 images
'Bacterial Canker' has 500 images
'Cutting Weevil' has 500 images
'Die Back' has 500 images
'Gall Midge' has 500 images
'Healthy' has 500 images
'Powdery Mildew' has 500 images
'Sooty Mould' has 500 images
```

Data Cleaning

- **Checking for null values**
- **Handling missing values**
- **Checking for duplicate values**

In [11]:

```
#Function for data cleaning
def data_cleaning(df, name='df'):
    #Checking for null values
    num_null_vals = sum(df.isnull().sum().values)

    #When there is no null values
    if not num_null_vals:
        print(f"The {name} has no null values")

    #When there i snull values
    else:
        print(f"The {name} has {num_null_vals} null values")
        print('Total null values in each column:\n')
        print(df.isnull().sum())

        #Removes rows with null values
        df = df.dropna()
        print(f"\nRows with null values have been removed. The dataset now has {d
f.shape[0]} rows.")

    #Checking for duplicates
    num_duplicates = df.duplicated().sum()

    #When there is no duplication in dataset
    if num_duplicates == 0:
        print(f"\nThe {name} has no duplicate values.")

    #When there is duplication in dataset
    else:
        print(f"\nThe {name} has {num_duplicates} duplicate rows.")
        df = df.drop_duplicates()
        print(f"Duplicate rows have been removed. The dataset now has {df.shape[0
]} rows.")

    return df

#Assiging new cleaned dataframe to the df
df = data_cleaning(df, dataset_name)
```

```
The Mango Leaf Disease Dataset has no null values

The Mango Leaf Disease Dataset has no duplicate values.
```

EDAs Class distribution in dataset

- **Shows the balance in dataset**

- **Helps model to be balanced and not biased towards any class**

In [12]:

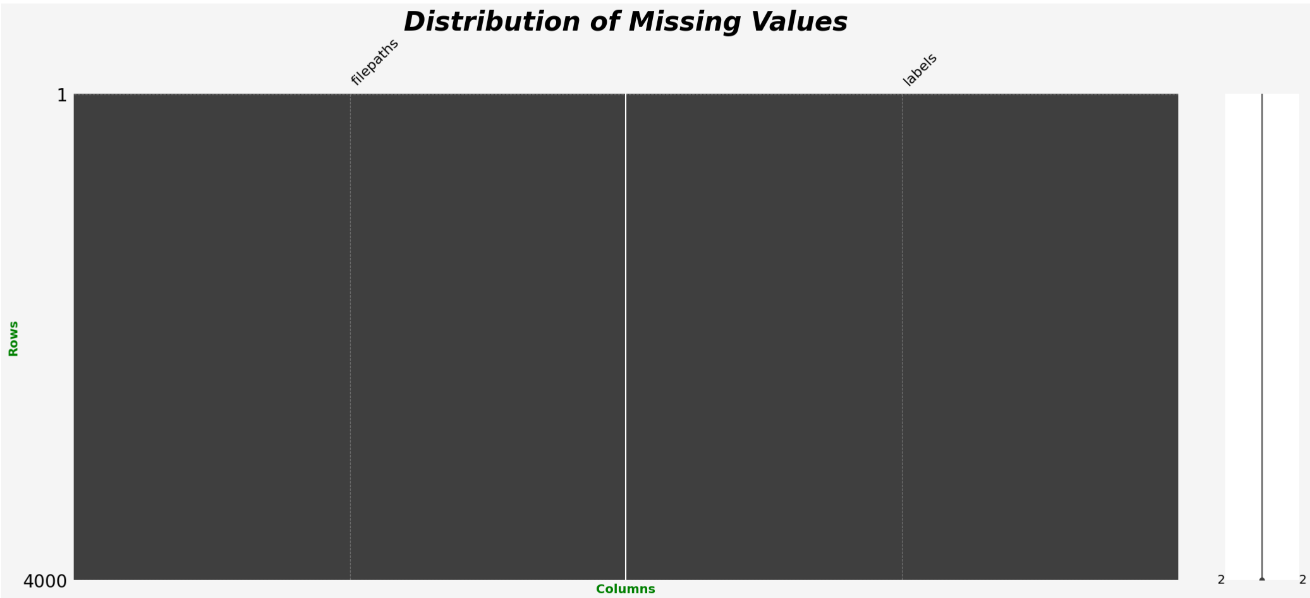
```
#Function to create graphs with class distribution in dataset
def class_distribution(dataframe, col_name):

    #Making subplots
    fig = make_subplots(rows=1, cols=2,
                        subplot_titles=('Percentage Plot', 'Total Count Plot'),
                        specs=[[{"type": "bar"}, {'type': 'scatter'}]])

    #Total counts in dataframe
    total_count = dataframe[col_name].value_counts().sum()
    #Percentage of particular label in dataframe
    percentage_values = (dataframe[col_name].value_counts().values / total_count)
* 100

    #Creating bar plot
    fig.add_trace(go.Bar(y=percentage_values.tolist(),
                        x=[str(i) for i in dataframe[col_name].value_counts().in
dex],
                        #Showing the values in percentage
                        text=[f'{val:.2f}%' for val in percentage_values],
                        textfont=dict(size=10),
                        name=col_name,
                        textposition='auto',
                        showlegend=False,
                        marker=dict(color=colors)),
                                )

    #Creating scatter plot
    fig.add_trace(go.Scatter(x=dataframe[col_name].value_counts().keys(),
                         y=dataframe[col_name].value_counts().values,
                         mode='markers',
                         text=dataframe[col_name].value_counts().keys(),
                         textfont=dict(size=10),
                         marker=dict(size=15, color=colors),
                         name=col_name),
              row=1, col=2)

    #Updating plot
    fig.update_layout(title={'text': 'Disease Distribution in Dataset',
                             'y': 0.9,
                             'x': 0.5,
                             'xanchor': 'center',
                             'yanchor': 'top'},
                      template='plotly_white')

    iplot(fig)


#Styling the plot with custom colours
colors = [
    '#3A506B',
    '#8E8D8A',
    '#D9BF77',
    '#6A8D73',
    '#B84A4A',
    '#86B3D1',
    '#B0C4B1',
    '#9A5A6E',
    '#C8A165',
    '#7C6C8E'
]

#Calling the function
```

```
class_distribution(df,'labels')
```

In [13]:

```
#Ploting the missing values matrix
msno.matrix(df)

#Setting Title and styles
plt.title('Distribution of Missing Values', fontsize=30, fontstyle='oblique', fon
tweight='bold')

#Custom fonts and colours
plt.xlabel('Columns', fontsize=14, fontweight='bold', color='green')
plt.ylabel('Rows', fontsize=14, fontweight='bold', color='green')

#Custom Background for the plot
plt.gcf().set_facecolor('whitesmoke')
plt.grid(True, linestyle='--', alpha=0.5)

#Adjusting of spacing layout
plt.tight_layout()

#Showing the plot
plt.show()
```

```
C:\Users\sudee\AppData\Local\Temp\ipykernel_17976\4089772672.py:16: UserWarning:

This figure includes Axes that are not compatible with tight_layout, so results mi
ght be incorrect.
```



Pixel Intensity Distribution

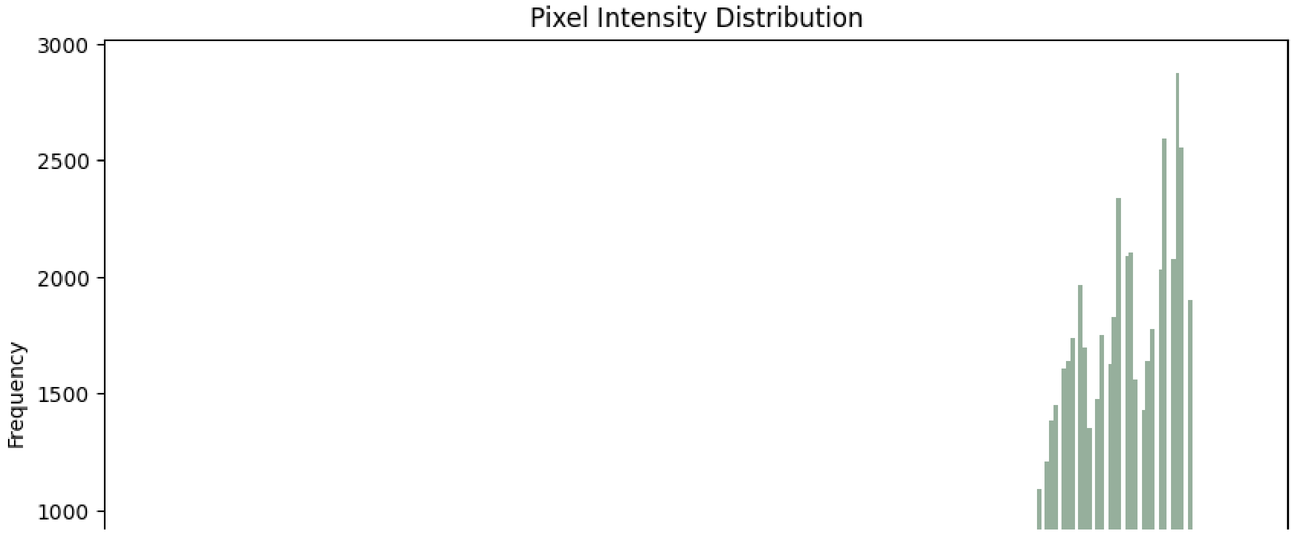
- **helps to understand the image better and see the features and patterns on the image**
- **helps to identify textures patterns**
- **help to differential objects in image**

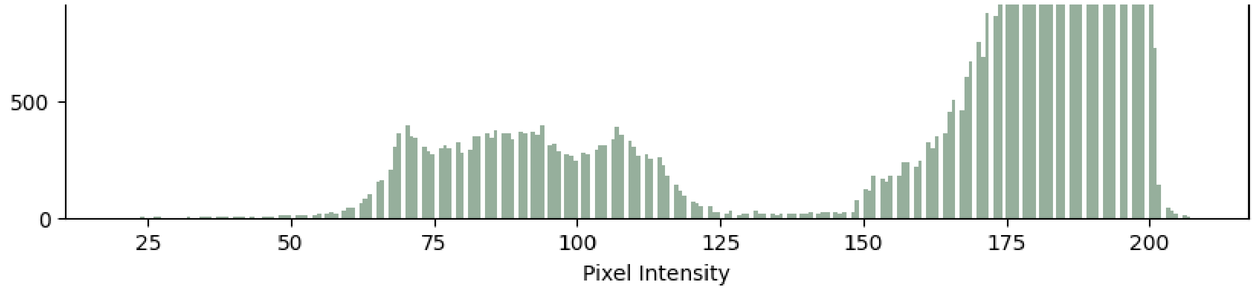
In [14]:

```
#Checking one image's pixel intensity and edge detection to see if its good

#Picking a random image from the DataFrame
ran_index = random.choice(df.index)
ran_filepath = df.loc[ran_index, 'filepaths']
ran_label = df.loc[ran_index, 'labels']
```

```python
#Loading the selected image in grayscale
img = cv2.imread(ran_filepath, cv2.IMREAD_GRAYSCALE)

#Checking if the image was loaded properly
if img is not None:
    print(f"Selected Image: {ran_filepath}, Label: {ran_label}")

    #Pixel Intensity Distribution (Histogram) plot
    plt.figure(figsize=(10, 6))

    #Flatting the image array
    plt.hist(img.ravel(), bins=256, color='#6A8D73', alpha=0.7)
    plt.title("Pixel Intensity Distribution")
    plt.xlabel("Pixel Intensity")
    plt.ylabel("Frequency")
    plt.show()

    #Basic Statistics of Pixel Intensities
    mean_intensity = np.mean(img)
    std_intensity = np.std(img)
    min_intensity = np.min(img)
    max_intensity = np.max(img)
    print(f"Image Statistics - Mean: {mean_intensity}, Standard Deviation: {std_i
ntensity}, Min: {min_intensity}, Max: {max_intensity}")

    #Displaying the  Grey scale Image
    plt.figure(figsize=(6, 6))
    plt.imshow(img, cmap='gray')
    plt.title(f"Image - Label: {ran_label}")
    plt.axis('off')
    plt.show()

    #Edge Detection Using Sobel Filter from opencv
    sobel_x = cv2.Sobel(img, cv2.CV_64F, 1, 0, ksize=3)
    sobel_y = cv2.Sobel(img, cv2.CV_64F, 0, 1, ksize=3)
    magnitude = cv2.magnitude(sobel_x, sobel_y)

    #Visualizing the Edge Detected Image
    plt.figure(figsize=(6, 6))
    plt.imshow(magnitude, cmap='hot')
    plt.title(f"Edge Detection with Sobel for Image - Label: {ran_label}")
    plt.axis('off')
    plt.show()

else:
    print("Error loading the image!")
```

```
Selected Image: D:/Big Data Analytics/Term-2/BDM 3014 - Introduction to Artificial
Intelligence 01/Final Project/MangoLeafBD Dataset\Anthracnose\IMG_20211011_161636
(Custom).jpg, Label: Anthracnose
```

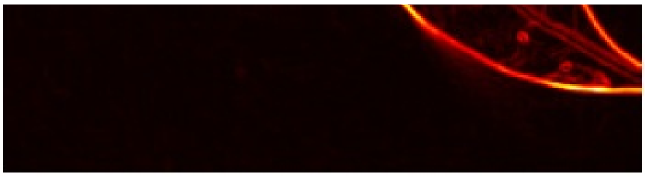





```
Image Statistics - Mean: 163.34067708333333, Standard Deviation: 41.45723834265726
, Min: 20, Max: 208
```

Image - Label: Anthracnose

Edge Detection with Sobel for Image - Label: Anthracnose



Train, Test, Validation Split

In [15]:

```
#Splitting data into training testing and validation
#Training dataframe
train_df, dummy_df = train_test_split(df,  train_size= 0.7, shuffle= True, random
_state= 123)

#validation and test dataframe
validation_df, test_df = train_test_split(dummy_df,  train_size= 0.5, shuffle= Tr
ue, random_state= 123)
```

In [16]:

```
#Function to check for size of datasets
def data_size(df, name='df'):
    print(f"Number of {name} is {len(df)} images")
```

In [17]:

```
#Training dataset size
data_size(train_df, 'Training '+dataset_name)

#Validation dataset size
data_size(validation_df, 'Validation '+dataset_name)

#Testing dataset size
data_size(test_df, 'Testing '+dataset_name)
```

```
Number of Training Mango Leaf Disease Dataset is 2800 images
Number of Validation Mango Leaf Disease Dataset is 600 images
Number of Testing Mango Leaf Disease Dataset is 600 images
```

Feature Engineering

- **Image Normalization and Scaling**
- **Image Augmentation (Rotations, Brightness, Flips)**
- **Handling Input Sizes and Channels**
- **Batch Processing**

In [18]:

```
#Defining image and batch size parameters

#Number of images to be processed in a batch
batch_size = 40
#Cropping image size (width, height) in pixels
img_size = (224, 224)
#Number of color channels (RGB)
channels = 3

#Shape of the input image
img_shape = (img_size[0], img_size[1], channels)

#Calculating custom test batch size based on test dataset length
ts_length = len(test_df)
```

```
#Finding the optimal test batch size where number of steps is <= 80
test_batch_size = max(sorted([ts_length // n for n in range(1, ts_length + 1) if
ts_length % n == 0 and ts_length / n <= 80]))

#Calculating the number of steps per epoch for the test dataset
test_steps = ts_length // test_batch_size

#Custom scalar function to be used in the ImageDataGenerator; it returns the image
without any changes
def scalar(img):
    return img

#Creating an ImageDataGenerator for training with data augmentation (rotation, shi
fting, zooming, flipping, etc.)
training_gen = ImageDataGenerator(preprocessing_function=scalar,  # Apply the sca
lar function to the images
                            #Data augmentation parameters
                            rotation_range=40,
                            width_shift_range=0.2,
                            height_shift_range=0.2,
                            brightness_range=[0.4, 0.6],
                            zoom_range=0.3,
                            horizontal_flip=True,
                            vertical_flip=True)

#Creating a similar ImageDataGenerator for testing (no data augmentation, just sca
lar function)
#Appling the scalar function to the images
testing_gen = ImageDataGenerator(preprocessing_function=scalar,
                            rotation_range=40,
                            width_shift_range=0.2,
                            height_shift_range=0.2,
                            brightness_range=[0.4, 0.6],
                            zoom_range=0.3,
                            horizontal_flip=True,
                            vertical_flip=True)

#Generating training data from a DataFrame
train_gen = training_gen.flow_from_dataframe(train_df,  #DataFrame with training
data paths and labels
                                       #Column name for image file paths
                                       x_col='filepaths',
                                       #Column name for image labels
                                       y_col='labels',
                                       #Resize images to target size (224x224)
                                       target_size=img_size,
                                       #Class mode for categorical labels (multi
-class classification)
                                       class_mode='categorical',
                                       #Load images as RGB (3 channels)
                                       color_mode='rgb',
                                       #Shuffle the data for better training
                                       shuffle=True,
                                       #Number of images per batch
                                       batch_size=batch_size)

#Generating validation data from a DataFrame
validation_gen = testing_gen.flow_from_dataframe(validation_df,  #DataFrame with
validation data paths and labels
                                            x_col='filepaths',
                                            y_col='labels',
                                            target_size=img_size,
                                            class_mode='categorical',
                                            color_mode='rgb',
                                            shuffle=True,  #Shuffle validation d
ata
                                            batch_size=batch_size)
```

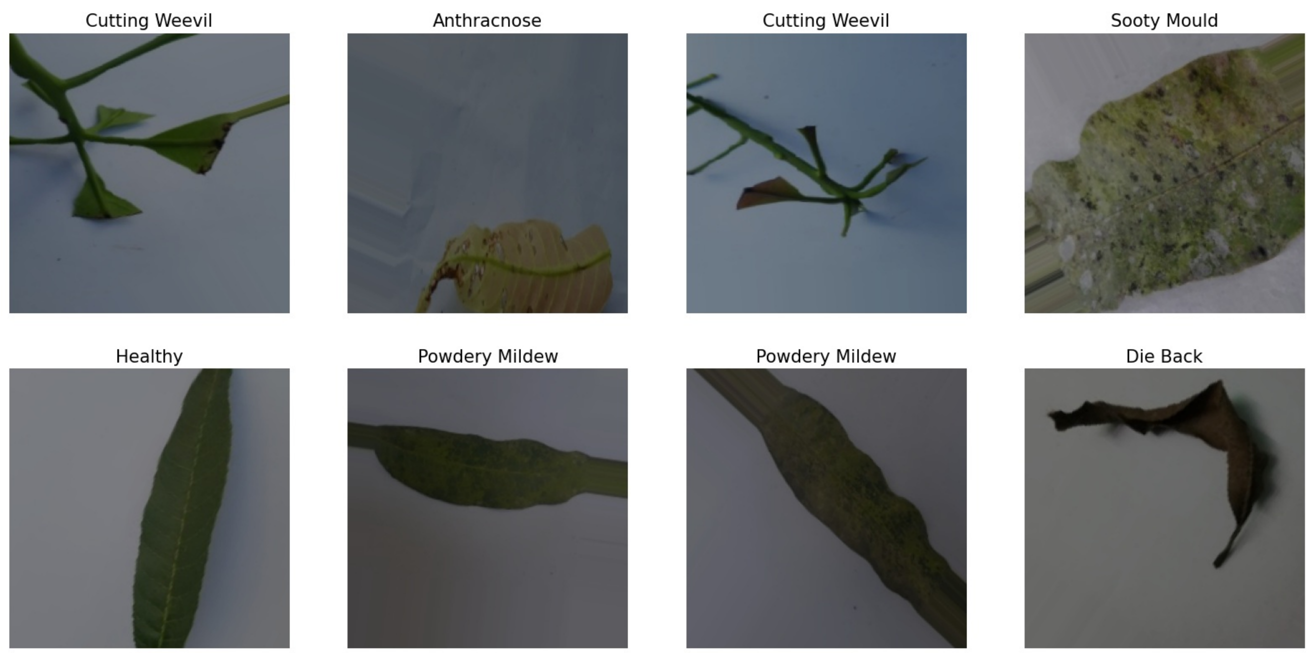
```python
#Generating test data from a DataFrame
#Using custom test_batch_size and no shuffling since the test data needs to be ev
aluated as it is
#DataFrame with test data paths and labels
test_gen = testing_gen.flow_from_dataframe(test_df,
                                          x_col='filepaths',
                                          y_col='labels',
                                          target_size=img_size,
                                          class_mode='categorical',
                                          color_mode='rgb',
                                          #Do not shuffle test data
                                          shuffle=False,
                                          #Custom test batch size calculated earlier
                                          batch_size=test_batch_size)
```

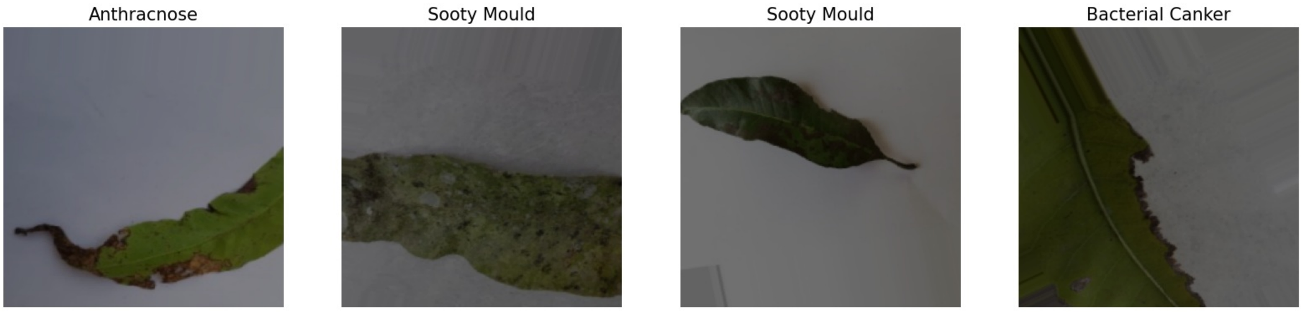
```
Found 2800 validated image filenames belonging to 8 classes.
Found 600 validated image filenames belonging to 8 classes.
Found 600 validated image filenames belonging to 8 classes.
```

Checking batch sample from training data

In [19]:

```python
#defines dictionary {'class': index}
gen_dict = train_gen.class_indices

#defines list of dictionary's kays (classes), classes names : string
classes = list(gen_dict.keys())

#get a batch size samples from the generator
images, labels = next(train_gen)

plt.figure(figsize= (20, 20))

for i in range(12):
    plt.subplot(4, 4, i + 1)
    #scaling data to range (0 - 255)
    image = images[i] / 255
    plt.imshow(image)
    index = np.argmax(labels[i])
    class_name = classes[index]
    plt.title(class_name, color= 'black', fontsize= 15)
    plt.axis('off')

plt.show()
```





Squential Model Building

In [17]:

```
#Creating Model Structure
img_size = (224, 224)
channels = 3
img_shape = (img_size[0], img_size[1], channels)

#to define number of classes in dense layer
class_count = len(list(train_gen.class_indices.keys()))


#using efficientnetb0 from EfficientNet family.
#Simpler base model istead of complex one,
#because small dataset and to stop overfitting

base_model = tf.keras.applications.efficientnet.EfficientNetB0(include_top= False
, weights= "imagenet", input_shape= img_shape, pooling= 'max')
base_model.trainable = False

#Building a Sequential model with the EfficientNetB7 base
model = Sequential([
    #Adding the base model
    base_model,

    #Normalizing inputs for faster training and convergence
    BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001),
    #Adding a fully connected layer with 128 units
    Dense(128,
          #Adding L2 regularization to the weights
          kernel_regularizer=regularizers.l2(0.01),
          #Adding L1 regularization to the activations
          activity_regularizer=regularizers.l1(0.001),
          #Adding L1 regularization to the biases
          bias_regularizer=regularizers.l1(0.001),
          #Using ReLU activation function
          activation='relu'),
    #Dropout layer to prevent overfitting with a dropout rate of 45%
    Dropout(rate=0.3, seed=123),

    #Output layer with softmax activation for multi-class classification
    Dense(class_count, activation='softmax')
])

#Compiling the model
#Adamax is an adaptive learning rate optimizer based on Adam
#categorical_crossentropy is used as the loss function for multi-class classifica
tion
#Using Adamax optimizer with learning rate 0.001
model.compile(optimizer=Adamax(learning_rate=0.0001),
               #Loss function for categorical classification
              loss='categorical_crossentropy',
              #Metric to monitor during training is accuracy
              metrics=['accuracy'])
```

```
#Displaying the model architecture summary
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
efficientnetb0 (Functional)	(None, 1280)	4,049,571
batch_normalization (BatchNormalization)	(None, 1280)	5,120
dense (Dense)	(None, 128)	163,968
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 8)	1,032

Total params: 4,219,691 (16.10 MB)

Trainable params: 167,560 (654.53 KB)

Non-trainable params: 4,052,131 (15.46 MB)

Putting Early Stop for training the data

In [18]:

```
#Setting up EarlyStopping callback
early_stopping = EarlyStopping(
    #Monitors the validation loss during training
    monitor='val_loss',
    #If validation loss doesn't improve for 5 consecutive epochs, training stops
early
    #Change is accordingly to the number of epochs you want to wait before stoppi
ng
    patience=5,
    #Restores the model's best weights (with the lowest validation loss) after st
opping
    restore_best_weights=True,
    #Looks for the minimum value of 'val_loss' (we want to minimize the loss)
    mode='min'
)
```

Data Training with epochs 10 for initial training

In [ ]:

```
#Setting Bacth size fir training
batch_size = 16

#Setting number of epochs for training
epochs = 100

history = model.fit(x=train_gen,
                    epochs = epochs,
                    verbose = 1,
                    validation_data = validation_gen,
                    validation_steps = None,
                    shuffle = False,
                    batch_size = batch_size,
```

```
                    callbacks = [early_stopping])
```

```
c:\Users\sudee\AppData\Local\Programs\Python\Python312\Lib\site-packages\keras\src
\trainers\data_adapters\py_dataset_adapter.py:121: UserWarning:

Your `PyDataset` class should call `super().__init__(**kwargs)` in its constructor
. `**kwargs` can include `workers`, `use_multiprocessing`, `max_queue_size`. Do no
t pass these arguments to `fit()`, as they will be ignored.
```

```
Epoch 1/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 178s 2s/step - accuracy: 0.2416 - loss: 7.4819 - val_a
ccuracy: 0.5133 - val_loss: 7.7311
Epoch 2/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 133s 2s/step - accuracy: 0.5067 - loss: 6.2678 - val_a
ccuracy: 0.7333 - val_loss: 6.1086
Epoch 3/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 124s 2s/step - accuracy: 0.6717 - loss: 5.7902 - val_a
ccuracy: 0.8000 - val_loss: 5.5382
Epoch 4/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 121s 2s/step - accuracy: 0.7323 - loss: 5.5009 - val_a
ccuracy: 0.8167 - val_loss: 5.2705
Epoch 5/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 112s 2s/step - accuracy: 0.7844 - loss: 5.2963 - val_a
ccuracy: 0.8650 - val_loss: 5.0180
Epoch 6/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 139s 2s/step - accuracy: 0.7988 - loss: 5.1093 - val_a
ccuracy: 0.8833 - val_loss: 4.8911
Epoch 7/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 142s 2s/step - accuracy: 0.8256 - loss: 4.9712 - val_a
ccuracy: 0.9100 - val_loss: 4.7237
Epoch 8/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 141s 2s/step - accuracy: 0.8367 - loss: 4.8391 - val_a
ccuracy: 0.9017 - val_loss: 4.6276
Epoch 9/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 142s 2s/step - accuracy: 0.8600 - loss: 4.7092 - val_a
ccuracy: 0.8900 - val_loss: 4.4970
Epoch 10/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 145s 2s/step - accuracy: 0.8736 - loss: 4.6020 - val_a
ccuracy: 0.9233 - val_loss: 4.3670
Epoch 11/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 142s 2s/step - accuracy: 0.8864 - loss: 4.4708 - val_a
ccuracy: 0.9217 - val_loss: 4.2850
Epoch 12/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 137s 2s/step - accuracy: 0.8764 - loss: 4.3744 - val_a
ccuracy: 0.9333 - val_loss: 4.1765
Epoch 13/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 138s 2s/step - accuracy: 0.8812 - loss: 4.2887 - val_a
ccuracy: 0.9233 - val_loss: 4.0670
Epoch 14/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 147s 2s/step - accuracy: 0.8974 - loss: 4.1513 - val_a
ccuracy: 0.9467 - val_loss: 3.9501
Epoch 15/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 141s 2s/step - accuracy: 0.8918 - loss: 4.0830 - val_a
ccuracy: 0.9267 - val_loss: 3.8730
Epoch 16/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 141s 2s/step - accuracy: 0.9066 - loss: 3.9435 - val_a
ccuracy: 0.9350 - val_loss: 3.8023
Epoch 17/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 139s 2s/step - accuracy: 0.9050 - loss: 3.8590 - val_a
ccuracy: 0.9517 - val_loss: 3.6703
Epoch 18/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 143s 2s/step - accuracy: 0.9096 - loss: 3.7600 - val_a
ccuracy: 0.9467 - val_loss: 3.6074
Epoch 19/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 139s 2s/step - accuracy: 0.9102 - loss: 3.6731 - val_a
ccuracy: 0.9400 - val_loss: 3.5005
Epoch 20/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 142s 2s/step - accuracy: 0.9168 - loss: 3.5785 - val_a
```

```
70/70 ━━━━━━━━━━━━━━━━━━━━ 142s 2s/step - accuracy: 0.9166 - loss: 3.5785 - val_a
ccuracy: 0.9500 - val_loss: 3.4331
Epoch 21/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 140s 2s/step - accuracy: 0.9294 - loss: 3.5041 - val_a
ccuracy: 0.9583 - val_loss: 3.3207
Epoch 22/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 140s 2s/step - accuracy: 0.9251 - loss: 3.4238 - val_a
ccuracy: 0.9400 - val_loss: 3.2836
Epoch 23/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 139s 2s/step - accuracy: 0.9202 - loss: 3.3481 - val_a
ccuracy: 0.9533 - val_loss: 3.1832
Epoch 24/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 142s 2s/step - accuracy: 0.9305 - loss: 3.2668 - val_a
ccuracy: 0.9567 - val_loss: 3.1270
Epoch 25/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 173s 2s/step - accuracy: 0.9348 - loss: 3.1916 - val_a
ccuracy: 0.9600 - val_loss: 3.0487
Epoch 26/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 146s 2s/step - accuracy: 0.9268 - loss: 3.1341 - val_a
ccuracy: 0.9517 - val_loss: 2.9957
Epoch 27/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 159s 2s/step - accuracy: 0.9411 - loss: 3.0563 - val_a
ccuracy: 0.9633 - val_loss: 2.9331
Epoch 28/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 139s 2s/step - accuracy: 0.9366 - loss: 2.9893 - val_a
ccuracy: 0.9550 - val_loss: 2.8815
Epoch 29/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 141s 2s/step - accuracy: 0.9350 - loss: 2.9445 - val_a
ccuracy: 0.9767 - val_loss: 2.8076
Epoch 30/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 136s 2s/step - accuracy: 0.9437 - loss: 2.8733 - val_a
ccuracy: 0.9617 - val_loss: 2.7595
Epoch 31/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 137s 2s/step - accuracy: 0.9422 - loss: 2.8296 - val_a
ccuracy: 0.9583 - val_loss: 2.7250
Epoch 32/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 137s 2s/step - accuracy: 0.9421 - loss: 2.7852 - val_a
ccuracy: 0.9583 - val_loss: 2.6674
Epoch 33/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 136s 2s/step - accuracy: 0.9423 - loss: 2.7339 - val_a
ccuracy: 0.9700 - val_loss: 2.6279
Epoch 34/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 135s 2s/step - accuracy: 0.9508 - loss: 2.6845 - val_a
ccuracy: 0.9700 - val_loss: 2.5841
Epoch 35/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 137s 2s/step - accuracy: 0.9518 - loss: 2.6310 - val_a
ccuracy: 0.9617 - val_loss: 2.5343
Epoch 36/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 135s 2s/step - accuracy: 0.9514 - loss: 2.5892 - val_a
ccuracy: 0.9550 - val_loss: 2.5016
Epoch 37/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 136s 2s/step - accuracy: 0.9565 - loss: 2.5748 - val_a
ccuracy: 0.9650 - val_loss: 2.4457
Epoch 38/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 134s 2s/step - accuracy: 0.9416 - loss: 2.5340 - val_a
ccuracy: 0.9650 - val_loss: 2.4199
Epoch 39/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 135s 2s/step - accuracy: 0.9561 - loss: 2.4709 - val_a
ccuracy: 0.9617 - val_loss: 2.3788
Epoch 40/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 135s 2s/step - accuracy: 0.9515 - loss: 2.4495 - val_a
ccuracy: 0.9717 - val_loss: 2.3378
Epoch 41/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 141s 2s/step - accuracy: 0.9563 - loss: 2.3938 - val_a
ccuracy: 0.9750 - val_loss: 2.2953
Epoch 42/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 182s 3s/step - accuracy: 0.9505 - loss: 2.3812 - val_a
ccuracy: 0.9700 - val_loss: 2.2665
Epoch 43/100
```

```
Epoch 43/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 157s 2s/step - accuracy: 0.9491 - loss: 2.3378 - val_a
ccuracy: 0.9750 - val_loss: 2.2304
Epoch 44/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 155s 2s/step - accuracy: 0.9622 - loss: 2.2917 - val_a
ccuracy: 0.9667 - val_loss: 2.2000
Epoch 45/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 160s 2s/step - accuracy: 0.9534 - loss: 2.2594 - val_a
ccuracy: 0.9733 - val_loss: 2.1761
Epoch 46/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 161s 2s/step - accuracy: 0.9554 - loss: 2.2365 - val_a
ccuracy: 0.9767 - val_loss: 2.1218
Epoch 47/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 180s 3s/step - accuracy: 0.9530 - loss: 2.2182 - val_a
ccuracy: 0.9700 - val_loss: 2.1048
Epoch 48/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 170s 2s/step - accuracy: 0.9561 - loss: 2.1792 - val_a
ccuracy: 0.9700 - val_loss: 2.0727
Epoch 49/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 183s 2s/step - accuracy: 0.9613 - loss: 2.1480 - val_a
ccuracy: 0.9717 - val_loss: 2.0534
Epoch 50/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 161s 2s/step - accuracy: 0.9613 - loss: 2.1105 - val_a
ccuracy: 0.9600 - val_loss: 2.0317
Epoch 51/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 163s 2s/step - accuracy: 0.9581 - loss: 2.0878 - val_a
ccuracy: 0.9717 - val_loss: 1.9902
Epoch 52/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 155s 2s/step - accuracy: 0.9579 - loss: 2.0708 - val_a
ccuracy: 0.9750 - val_loss: 1.9670
Epoch 53/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 158s 2s/step - accuracy: 0.9568 - loss: 2.0397 - val_a
ccuracy: 0.9650 - val_loss: 1.9489
Epoch 54/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 155s 2s/step - accuracy: 0.9611 - loss: 2.0039 - val_a
ccuracy: 0.9683 - val_loss: 1.9244
Epoch 55/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 149s 2s/step - accuracy: 0.9608 - loss: 1.9760 - val_a
ccuracy: 0.9767 - val_loss: 1.8813
Epoch 56/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 141s 2s/step - accuracy: 0.9579 - loss: 1.9593 - val_a
ccuracy: 0.9717 - val_loss: 1.8490
Epoch 57/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 138s 2s/step - accuracy: 0.9649 - loss: 1.9277 - val_a
ccuracy: 0.9767 - val_loss: 1.8155
Epoch 58/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 143s 2s/step - accuracy: 0.9694 - loss: 1.8935 - val_a
ccuracy: 0.9800 - val_loss: 1.7980
Epoch 59/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 149s 2s/step - accuracy: 0.9693 - loss: 1.8690 - val_a
ccuracy: 0.9717 - val_loss: 1.7769
Epoch 60/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 144s 2s/step - accuracy: 0.9656 - loss: 1.8648 - val_a
ccuracy: 0.9767 - val_loss: 1.7668
Epoch 61/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 134s 2s/step - accuracy: 0.9547 - loss: 1.8547 - val_a
ccuracy: 0.9767 - val_loss: 1.7361
Epoch 62/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 135s 2s/step - accuracy: 0.9622 - loss: 1.8037 - val_a
ccuracy: 0.9650 - val_loss: 1.7182
Epoch 63/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 134s 2s/step - accuracy: 0.9702 - loss: 1.7816 - val_a
ccuracy: 0.9783 - val_loss: 1.6880
Epoch 64/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 109s 2s/step - accuracy: 0.9729 - loss: 1.7450 - val_a
ccuracy: 0.9717 - val_loss: 1.6726
Epoch 65/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 108s 2s/step - accuracy: 0.9637 - loss: 1.7209 - val_a
ccuracy: 0.9700 - val_loss: 1.6452
```

```
ccuracy: 0.9700 - val_loss: 1.6432
Epoch 66/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 103s 1s/step - accuracy: 0.9601 - loss: 1.7320 - val_a
ccuracy: 0.9800 - val_loss: 1.6183
Epoch 67/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 108s 2s/step - accuracy: 0.9698 - loss: 1.6831 - val_a
ccuracy: 0.9867 - val_loss: 1.5888
Epoch 68/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 112s 2s/step - accuracy: 0.9699 - loss: 1.6711 - val_a
ccuracy: 0.9883 - val_loss: 1.5622
Epoch 69/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 105s 1s/step - accuracy: 0.9740 - loss: 1.6551 - val_a
ccuracy: 0.9833 - val_loss: 1.5631
Epoch 70/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 104s 1s/step - accuracy: 0.9681 - loss: 1.6381 - val_a
ccuracy: 0.9733 - val_loss: 1.5401
Epoch 71/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 107s 2s/step - accuracy: 0.9648 - loss: 1.6174 - val_a
ccuracy: 0.9833 - val_loss: 1.5117
Epoch 72/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 109s 2s/step - accuracy: 0.9770 - loss: 1.5858 - val_a
ccuracy: 0.9817 - val_loss: 1.4965
Epoch 73/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 107s 2s/step - accuracy: 0.9746 - loss: 1.5737 - val_a
ccuracy: 0.9800 - val_loss: 1.4732
Epoch 74/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 107s 2s/step - accuracy: 0.9728 - loss: 1.5527 - val_a
ccuracy: 0.9800 - val_loss: 1.4591
Epoch 75/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 121s 2s/step - accuracy: 0.9781 - loss: 1.5325 - val_a
ccuracy: 0.9850 - val_loss: 1.4371
Epoch 76/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 170s 2s/step - accuracy: 0.9714 - loss: 1.5163 - val_a
ccuracy: 0.9850 - val_loss: 1.4250
Epoch 77/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 132s 2s/step - accuracy: 0.9735 - loss: 1.5025 - val_a
ccuracy: 0.9850 - val_loss: 1.4020
Epoch 78/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 120s 2s/step - accuracy: 0.9660 - loss: 1.4752 - val_a
ccuracy: 0.9800 - val_loss: 1.3932
Epoch 79/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 139s 2s/step - accuracy: 0.9742 - loss: 1.4580 - val_a
ccuracy: 0.9867 - val_loss: 1.3720
Epoch 80/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 156s 2s/step - accuracy: 0.9745 - loss: 1.4391 - val_a
ccuracy: 0.9750 - val_loss: 1.3646
Epoch 81/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 150s 2s/step - accuracy: 0.9805 - loss: 1.4231 - val_a
ccuracy: 0.9867 - val_loss: 1.3310
Epoch 82/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 141s 2s/step - accuracy: 0.9764 - loss: 1.4124 - val_a
ccuracy: 0.9883 - val_loss: 1.3228
Epoch 83/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 149s 2s/step - accuracy: 0.9711 - loss: 1.3998 - val_a
ccuracy: 0.9817 - val_loss: 1.3181
Epoch 84/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 142s 2s/step - accuracy: 0.9704 - loss: 1.3929 - val_a
ccuracy: 0.9833 - val_loss: 1.2931
Epoch 85/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 149s 2s/step - accuracy: 0.9761 - loss: 1.3589 - val_a
ccuracy: 0.9850 - val_loss: 1.2724
Epoch 86/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 138s 2s/step - accuracy: 0.9823 - loss: 1.3446 - val_a
ccuracy: 0.9917 - val_loss: 1.2567
Epoch 87/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 116s 2s/step - accuracy: 0.9760 - loss: 1.3440 - val_a
ccuracy: 0.9917 - val_loss: 1.2470
Epoch 88/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 109s 2s/step - accuracy: 0.9782 - loss: 1.3113 - val_a
```

```
ccuracy: 0.9817 - val_loss: 1.2368
Epoch 89/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 117s 2s/step - accuracy: 0.9788 - loss: 1.2986 - val_a
ccuracy: 0.9867 - val_loss: 1.2222
Epoch 90/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 110s 2s/step - accuracy: 0.9756 - loss: 1.2945 - val_a
ccuracy: 0.9800 - val_loss: 1.2124
Epoch 91/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 109s 2s/step - accuracy: 0.9759 - loss: 1.2769 - val_a
ccuracy: 0.9883 - val_loss: 1.1857
Epoch 92/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 108s 2s/step - accuracy: 0.9763 - loss: 1.2714 - val_a
ccuracy: 0.9883 - val_loss: 1.1862
Epoch 93/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 114s 2s/step - accuracy: 0.9783 - loss: 1.2525 - val_a
ccuracy: 0.9850 - val_loss: 1.1745
Epoch 94/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 124s 2s/step - accuracy: 0.9771 - loss: 1.2383 - val_a
ccuracy: 0.9817 - val_loss: 1.1574
Epoch 95/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 113s 2s/step - accuracy: 0.9841 - loss: 1.2087 - val_a
ccuracy: 0.9867 - val_loss: 1.1499
Epoch 96/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 110s 2s/step - accuracy: 0.9760 - loss: 1.2174 - val_a
ccuracy: 0.9883 - val_loss: 1.1352
Epoch 97/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 107s 2s/step - accuracy: 0.9826 - loss: 1.2000 - val_a
ccuracy: 0.9917 - val_loss: 1.1147
Epoch 98/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 135s 2s/step - accuracy: 0.9765 - loss: 1.1955 - val_a
ccuracy: 0.9850 - val_loss: 1.1038
Epoch 99/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 134s 2s/step - accuracy: 0.9743 - loss: 1.1876 - val_a
ccuracy: 0.9850 - val_loss: 1.0939
Epoch 100/100
70/70 ━━━━━━━━━━━━━━━━━━━━ 109s 2s/step - accuracy: 0.9820 - loss: 1.1737 - val_a
ccuracy: 0.9800 - val_loss: 1.0972
```

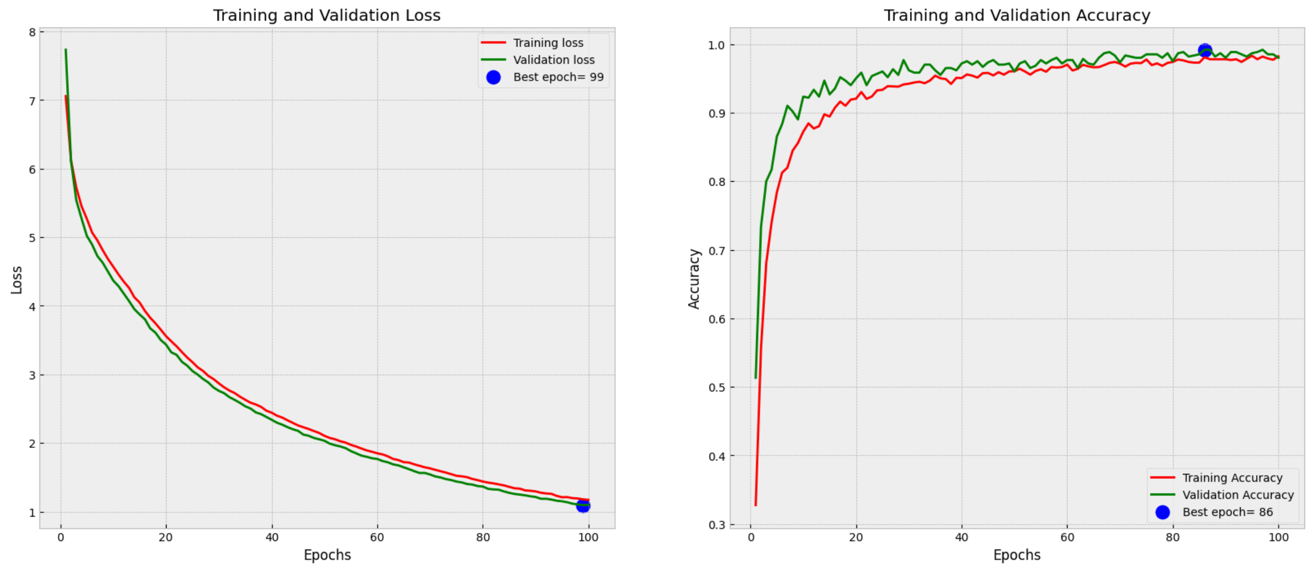
Model Evaluation plots

- **Training and Validation Loss**
- **Training and Validation Accuracy**

In [ ]:

```python
#Define needed variables
training_acc = history.history['accuracy']
training_loss = history.history['loss']
validation_acc = history.history['val_accuracy']
validation_loss = history.history['val_loss']
index_loss = np.argmin(validation_loss)
validation_lowest = validation_loss[index_loss]
index_acc = np.argmax(validation_acc)
acc_highest = validation_acc[index_acc]
Epochs = [i+1 for i in range(len(training_acc))]
loss_label = f'Best epoch= {str(index_loss + 1)}'
acc_label = f'Best epoch= {str(index_acc + 1)}'

#Plotting training history

plt.figure(figsize= (20, 8))
plt.style.use('bmh')

plt.subplot(1, 2, 1)
plt.plot(Epochs, training_loss, 'r', label= 'Training loss')
plt.plot(Epochs, validation_loss, 'g', label= 'Validation loss')
```

```
plt.scatter(index_loss + 1, validation_lowest, s= 150, c= 'blue', label= loss_lab
el)
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(Epochs, training_acc, 'r', label= 'Training Accuracy')
plt.plot(Epochs, validation_acc, 'g', label= 'Validation Accuracy')
plt.scatter(index_acc + 1 , acc_highest, s= 150, c= 'blue', label= acc_label)
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.tight_layout
plt.show()
```



Loss and Accuracy:

- **Training**
- **Validation**
- **Testing**

In [21]:

```
test_length = len(test_df)
test_batch_size = max(sorted([test_length // n for n in range(1, test_length + 1)
if test_length%n == 0 and test_length/n <= 80]))
test_steps = test_length // test_batch_size

train_score = model.evaluate(train_gen, steps= test_steps, verbose= 1)
valid_score = model.evaluate(validation_gen, steps= test_steps, verbose= 1)
test_score = model.evaluate(test_gen, steps= test_steps, verbose= 1)

print("Train Loss: ", train_score[0])
print("Train Accuracy: ", train_score[1])
print('-' * 20)
print("Validation Loss: ", valid_score[0])
print("Validation Accuracy: ", valid_score[1])
print('-' * 20)
print("Test Loss: ", test_score[0])
print("Test Accuracy: ", test_score[1])
```

```
8/8 ━━━━━━━━━━━━━━━━━━━━ 11s 1s/step - accuracy: 0.9979 - loss: 1.0685
8/8 ━━━━━━━━━━━━━━━━━━━━ 11s 1s/step - accuracy: 0.9956 - loss: 1.0841
```

```
c:\Users\sudee\AppData\Local\Programs\Python\Python312\Lib\site-packages\keras\src
\trainers\data_adapters\py_dataset_adapter.py:121: UserWarning:

Your `PyDataset` class should call `super().__init__(**kwargs)` in its constructor
. `**kwargs` can include `workers`, `use_multiprocessing`, `max_queue_size`. Do no
t pass these arguments to `fit()`, as they will be ignored.
```

```
8/8 ━━━━━━━━━━━━━━━━━━━━ 26s 3s/step - accuracy: 0.9877 - loss: 1.4885
Train Loss:  1.066455364227295
Train Accuracy:  0.996874988079071
--------------------
Validation Loss:  1.08376145362854
Validation Accuracy:  0.9937499761581421
--------------------
Test Loss:  1.4974197149276733
Test Accuracy:  0.9850000143051147
```

Prediction for Test

In [22]:

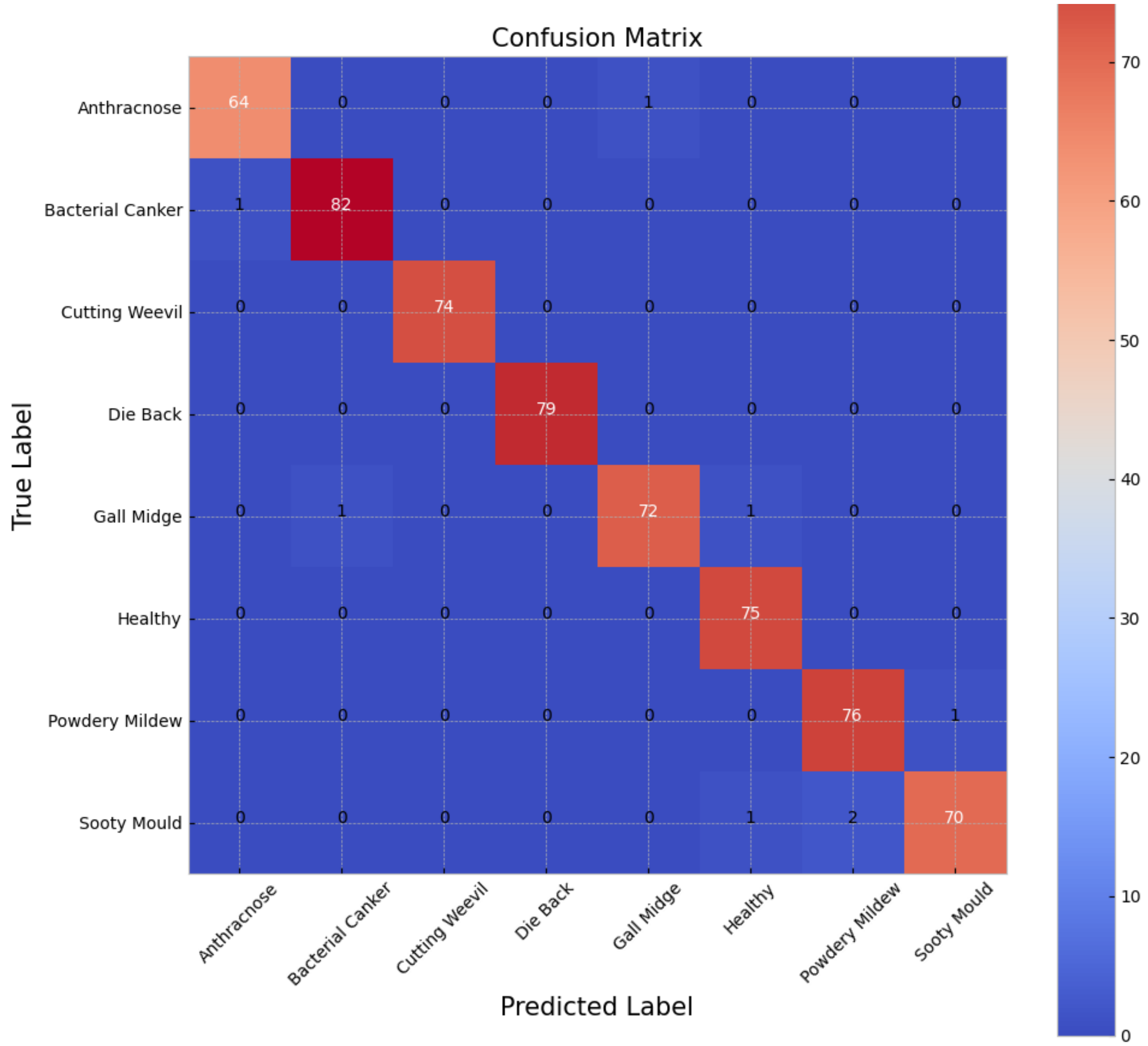
```
preds = model.predict(test_gen)
y_pred = np.argmax(preds, axis=1)
```

```
8/8 ━━━━━━━━━━━━━━━━━━━━ 27s 3s/step
```

Confustion Matrix for Prediction

In [23]:

```
g_dict = test_gen.class_indices
classes = list(g_dict.keys())

# Confusion matrix
cm = confusion_matrix(test_gen.classes, y_pred)

plt.figure(figsize= (10, 10))

#Picking plot style
plt.style.use('bmh')

#Colour theme
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.coolwarm)
plt.title('Confusion Matrix', fontsize=15)
plt.colorbar()

tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, classes, rotation= 45, fontsize=10)
plt.yticks(tick_marks, classes, fontsize=10)


thresh = cm.max() / 2.
for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
    plt.text(j, i, cm[i, j], horizontalalignment= 'center', color= 'white' if cm[
i, j] > thresh else 'black')
plt.tight_layout()
plt.ylabel('True Label', fontsize=15)
plt.xlabel('Predicted Label', fontsize=15)

plt.show()
```



Model Evaluation and Classification Report

- **Precision**
- **Recall**
- **F1-score**
- **Support**
- **Accuracy**
- **Macro Average**
- **Weighted Average**

In [24]:

```
#Printing classification report
print(classification_report(test_gen.classes, y_pred, target_names= classes))
```

```
                  precision    recall  f1-score   support

     Anthracnose       0.98      0.98      0.98        65
Bacterial Canker       0.99      0.99      0.99        83
  Cutting Weevil       1.00      1.00      1.00        74
        Die Back       1.00      1.00      1.00        79
      Gall Midge       0.99      0.97      0.98        74
         Healthy       0.97      1.00      0.99        75
  Powdery Mildew       0.97      0.99      0.98        77
     Sooty Mould       0.99      0.96      0.97        73
```

```
    accuracy                           0.99       600
   macro avg       0.99      0.99      0.99       600
weighted avg       0.99      0.99      0.99       600
```

Saving Model

In [ ]:

```
#Saving the model
model.save('model_v2.h5')
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `kera
s.saving.save_model(model)`. This file format is considered legacy. We recommend u
sing instead the native Keras format, e.g. `model.save('my_model.keras')` or `kera
s.saving.save_model(model, 'my_model.keras')`.
```

Model Interpretation With LIME

In [ ]:

```
from lime.lime_image import LimeImageExplainer

#Loading the saved model
model = load_model('model_v2.h5')

#Loading and preprocessing the image for model
img_path = 'D:/Big Data Analytics/Term-2/BDM 3014 - Introduction to Artificial In
telligence 01/Final Project/20211011_133423 (Custom).jpg'
#Resizing image to match input size
img = image.load_img(img_path, target_size=(224, 224))
img_array = image.img_to_array(img)
#Adding batch dimension
img_array = np.expand_dims(img_array, axis=0)

#Preprocessing image as required by EfficientNetB0
img_array_preprocessed = tf.keras.applications.efficientnet.preprocess_input(img_
array)

#Creating a LIME image explainer
explainer = LimeImageExplainer()

#Generating explanation using LIME
explanation = explainer.explain_instance(
    #Input image
    img_array[0],
    #Prediction  with model
    model.predict,
    #Number of top labels to explain (for classification)
    top_labels=5,
    #Color to hide (use 0 for default)
    hide_color=0,
    #Number of random samples to generate for explanation
    num_samples=1000
)

#Visualizing the explanation as an image overlay (heatmap)
#Creating two subplots side by side with original image and LIME explainer image
fig, axes = plt.subplots(1, 2, figsize=(15, 10))

#Original Image
axes[0].imshow(np.array(img))
axes[0].set_title('Original Image')
axes[0].axis('off')

#LIME Explanation Heatmap
```

```
temp, mask = explanation.get_image_and_mask(
    explanation.top_labels[0],
    positive_only=True,
    #Number of important features to highlight
    num_features=10,
    hide_rest=True
)
#Displaying the original image (for context)
axes[1].imshow(temp)
#Overlayying the heatmap with plasma colormap
heatmap = axes[1].imshow(mask, cmap='plasma', alpha=0.5)
axes[1].set_title('LIME Explanation: Heatmap of Important Features')
axes[1].axis('off')

#Adding color bar (legend)
cbar = fig.colorbar(heatmap, ax=axes[1], orientation='vertical')
cbar.set_label('Importance', rotation=270, labelpad=15)

#Printing the LIME explanation details
print("\nLIME Explanation Results:")
print("Top Label for Explanation:", explanation.top_labels[0])
print("Explanation of Important Features:")
for i, (feature, weight) in enumerate(explanation.local_exp[explanation.top_label
s[0]]):
    print(f"Feature {i+1}: {feature} with weight: {weight:.4f}")

#Showing the plot
#Adjusting layout to prevent overlay
plt.tight_layout()
plt.show()
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be bu
ilt. `model.compile_metrics` will be empty until you train or evaluate the model.
  0%|          | 0/1000 [00:00<?, ?it/s]

1/1 ━━━━━━━━━━━━━━━━━━━━ 4s 4s/step

  1%|          | 10/1000 [00:04<06:39,  2.48it/s]

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 355ms/step

  2%|▏         | 20/1000 [00:04<03:13,  5.08it/s]

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 458ms/step

  3%|▎         | 30/1000 [00:05<02:09,  7.49it/s]

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 334ms/step

  4%|▍         | 40/1000 [00:05<01:36,  9.96it/s]

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 288ms/step

  5%|▌         | 50/1000 [00:06<01:14, 12.82it/s]

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 283ms/step

  6%|▌         | 60/1000 [00:06<00:59, 15.67it/s]

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 282ms/step

  7%|▋         | 70/1000 [00:06<00:50, 18.31it/s]

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 287ms/step

  8%|▊         | 80/1000 [00:07<00:44, 20.50it/s]

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 274ms/step

  9%|▉         | 90/1000 [00:07<00:40, 22.23it/s]

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 277ms/step
```

```
 10%|█         | 100/1000 [00:07<00:37, 23.82it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 277ms/step
 11%|█         | 110/1000 [00:08<00:35, 24.85it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 310ms/step
 12%|█▏        | 120/1000 [00:08<00:34, 25.15it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 320ms/step
 13%|█▎        | 130/1000 [00:09<00:36, 23.70it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 312ms/step
 14%|█▍        | 140/1000 [00:09<00:35, 24.07it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 333ms/step
 15%|█▌        | 150/1000 [00:09<00:35, 23.99it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 322ms/step
 16%|█▌        | 160/1000 [00:10<00:35, 23.77it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 287ms/step
 17%|█▋        | 170/1000 [00:10<00:33, 24.50it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 284ms/step
 18%|█▊        | 180/1000 [00:11<00:32, 25.40it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 288ms/step
 19%|█▉        | 190/1000 [00:11<00:31, 26.01it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 276ms/step
 20%|██        | 200/1000 [00:11<00:29, 26.68it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 287ms/step
 21%|██        | 210/1000 [00:12<00:29, 26.97it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 302ms/step
 22%|██▏       | 220/1000 [00:12<00:28, 26.93it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 321ms/step
 23%|██▎       | 230/1000 [00:12<00:29, 26.09it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 317ms/step
 24%|██▍       | 240/1000 [00:13<00:30, 24.78it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 331ms/step
 25%|██▌       | 250/1000 [00:13<00:30, 24.42it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 318ms/step
 26%|██▌       | 260/1000 [00:14<00:30, 24.52it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 325ms/step
 27%|██▋       | 270/1000 [00:14<00:29, 24.57it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 323ms/step
 28%|██▊       | 280/1000 [00:14<00:29, 24.74it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 332ms/step
```

```
 29%|██▉       | 290/1000 [00:15<00:28, 24.74it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 355ms/step
 30%|███       | 300/1000 [00:15<00:29, 23.87it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 355ms/step
 31%|███       | 310/1000 [00:16<00:29, 23.34it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 318ms/step
 32%|███▏      | 320/1000 [00:16<00:29, 23.36it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 311ms/step
 33%|███▎      | 330/1000 [00:17<00:28, 23.83it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 318ms/step
 34%|███▍      | 340/1000 [00:17<00:27, 23.60it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 315ms/step
 35%|███▌      | 350/1000 [00:17<00:27, 24.00it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 313ms/step
 36%|███▌      | 360/1000 [00:18<00:26, 24.30it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 321ms/step
 37%|███▋      | 370/1000 [00:18<00:26, 23.54it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 289ms/step
 38%|███▊      | 380/1000 [00:19<00:25, 24.06it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 295ms/step
 39%|███▉      | 390/1000 [00:19<00:25, 24.34it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 303ms/step
 40%|████      | 400/1000 [00:19<00:24, 24.83it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 308ms/step
 41%|████      | 410/1000 [00:20<00:23, 24.83it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 312ms/step
 42%|████▏     | 420/1000 [00:20<00:23, 24.91it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 305ms/step
 43%|████▎     | 430/1000 [00:21<00:22, 24.97it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 307ms/step
 44%|████▍     | 440/1000 [00:21<00:22, 24.95it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 313ms/step
 45%|████▌     | 450/1000 [00:21<00:22, 24.96it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 353ms/step
 46%|████▌     | 460/1000 [00:22<00:22, 23.97it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 302ms/step
 47%|████▋     | 470/1000 [00:22<00:21, 24.11it/s]
```

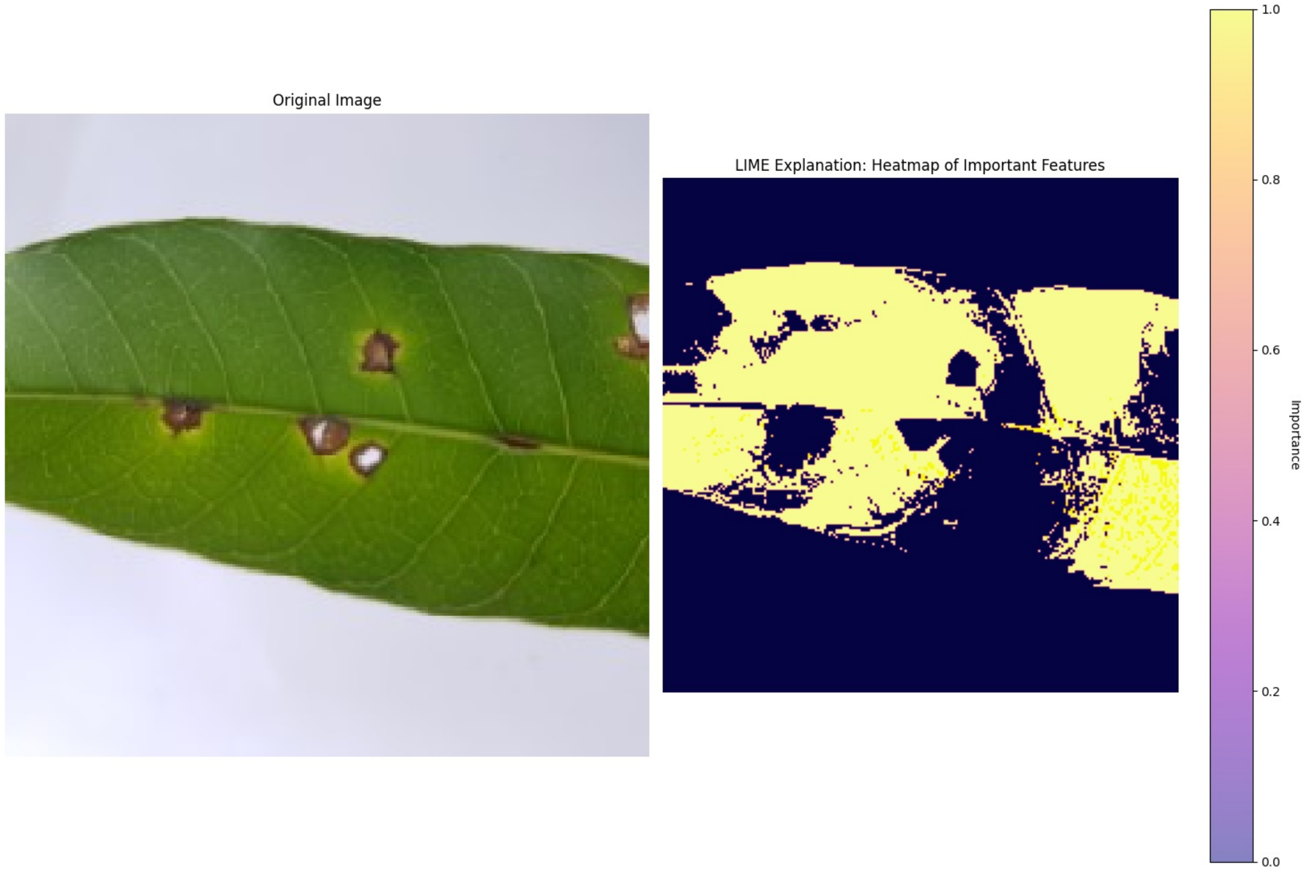
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 319ms/step
 48%|████▊     | 480/1000 [00:23<00:21, 23.79it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 316ms/step
 49%|████▉     | 490/1000 [00:23<00:21, 23.58it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 307ms/step
 50%|█████     | 500/1000 [00:24<00:21, 23.05it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 317ms/step
 51%|█████     | 510/1000 [00:24<00:20, 23.37it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 318ms/step
 52%|█████▏    | 520/1000 [00:24<00:20, 23.84it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 320ms/step
 54%|█████▍    | 538/1000 [00:25<00:15, 29.22it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 321ms/step
 54%|█████▍    | 542/1000 [00:25<00:20, 22.00it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 323ms/step
 55%|█████▌    | 550/1000 [00:26<00:21, 21.35it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 315ms/step
 56%|█████▌    | 560/1000 [00:26<00:19, 22.30it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 312ms/step
 57%|█████▋    | 570/1000 [00:27<00:18, 23.15it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 309ms/step
 58%|█████▊    | 580/1000 [00:27<00:17, 23.69it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 307ms/step
 59%|█████▉    | 590/1000 [00:27<00:17, 24.10it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 300ms/step
 60%|██████    | 600/1000 [00:28<00:17, 23.16it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 306ms/step
 61%|██████    | 610/1000 [00:28<00:16, 23.96it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 307ms/step
 62%|██████▏   | 620/1000 [00:29<00:15, 24.25it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 302ms/step
 63%|██████▎   | 630/1000 [00:29<00:15, 24.29it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 304ms/step
 64%|██████▍   | 640/1000 [00:29<00:14, 24.67it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 308ms/step
 65%|██████▌   | 650/1000 [00:30<00:14, 24.52it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 308ms/step
 66%|██████▌   | 660/1000 [00:30<00:13, 24.90it/s]
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 309ms/step
 67%|██████▋   | 670/1000 [00:31<00:13, 24.92it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 310ms/step
 68%|██████▊   | 680/1000 [00:31<00:12, 24.93it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 306ms/step
 69%|██████▉   | 690/1000 [00:31<00:12, 24.94it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 311ms/step
 70%|███████   | 700/1000 [00:32<00:12, 24.77it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 310ms/step
 71%|███████   | 710/1000 [00:32<00:11, 24.91it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 320ms/step
 72%|███████▏  | 720/1000 [00:33<00:11, 24.01it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 313ms/step
 73%|███████▎  | 730/1000 [00:33<00:11, 23.17it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 311ms/step
 74%|███████▍  | 740/1000 [00:34<00:10, 23.64it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 306ms/step
 75%|███████▌  | 750/1000 [00:34<00:10, 23.79it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 306ms/step
 76%|███████▌  | 760/1000 [00:34<00:09, 24.12it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 326ms/step
 77%|███████▋  | 770/1000 [00:35<00:09, 24.08it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 307ms/step
 78%|███████▊  | 780/1000 [00:35<00:09, 24.42it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 303ms/step
 79%|███████▉  | 790/1000 [00:36<00:08, 24.23it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 308ms/step
 80%|████████  | 800/1000 [00:36<00:08, 24.44it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 307ms/step
 81%|████████  | 810/1000 [00:36<00:07, 24.32it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 306ms/step
 82%|████████▏ | 820/1000 [00:37<00:07, 24.59it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 313ms/step
 83%|████████▎ | 830/1000 [00:37<00:06, 24.63it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 305ms/step
 84%|████████▍ | 840/1000 [00:38<00:06, 24.52it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 308ms/step
 86%|████████▌ | 859/1000 [00:38<00:04, 30.39it/s]
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 307ms/step
86%|████████▌ | 864/1000 [00:39<00:05, 24.22it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 306ms/step
87%|████████▋ | 870/1000 [00:39<00:06, 21.12it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 297ms/step
88%|████████▊ | 880/1000 [00:39<00:05, 22.37it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 322ms/step
89%|████████▉ | 890/1000 [00:40<00:04, 22.90it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 306ms/step
90%|█████████ | 900/1000 [00:40<00:04, 23.51it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 299ms/step
91%|█████████ | 910/1000 [00:41<00:03, 23.98it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 302ms/step
92%|█████████▏| 920/1000 [00:41<00:03, 23.80it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 331ms/step
93%|█████████▎| 930/1000 [00:41<00:02, 23.81it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 315ms/step
94%|█████████▍| 940/1000 [00:42<00:02, 23.30it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 311ms/step
95%|█████████▌| 950/1000 [00:42<00:02, 23.77it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 310ms/step
96%|█████████▌| 960/1000 [00:43<00:01, 23.91it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 305ms/step
97%|█████████▋| 970/1000 [00:43<00:01, 24.14it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 326ms/step
98%|█████████▊| 980/1000 [00:44<00:00, 22.53it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 349ms/step
99%|█████████▉| 990/1000 [00:44<00:00, 22.21it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 400ms/step
100%|██████████| 1000/1000 [00:45<00:00, 22.15it/s]
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RG
B data ([0..1] for floats or [0..255] for integers). Got range [0.0..174.0].

LIME Explanation Results:
Top Label for Explanation: 0
Explanation of Important Features:
Feature 1: 24 with weight: 0.0944
Feature 2: 67 with weight: 0.0834
Feature 3: 41 with weight: 0.0828
Feature 4: 37 with weight: 0.0738
Feature 5: 75 with weight: 0.0616
Feature 6: 27 with weight: 0.0567
Feature 7: 28 with weight: 0.0459
Feature 8: 40 with weight: 0.0457
Feature 9: 21 with weight: 0.0339
```

```
Feature 10: 65 with weight: 0.0333
Feature 11: 71 with weight: 0.0328
Feature 12: 77 with weight: 0.0322
Feature 13: 62 with weight: 0.0289
Feature 14: 11 with weight: 0.0277
Feature 15: 9 with weight: 0.0260
Feature 16: 43 with weight: 0.0259
Feature 17: 88 with weight: -0.0246
Feature 18: 66 with weight: 0.0245
Feature 19: 56 with weight: 0.0243
Feature 20: 23 with weight: 0.0196
Feature 21: 31 with weight: -0.0190
Feature 22: 26 with weight: 0.0178
Feature 23: 58 with weight: 0.0174
Feature 24: 64 with weight: 0.0172
Feature 25: 84 with weight: -0.0164
Feature 26: 53 with weight: 0.0162
Feature 27: 90 with weight: -0.0162
Feature 28: 82 with weight: 0.0157
Feature 29: 52 with weight: 0.0151
Feature 30: 30 with weight: -0.0144
Feature 31: 86 with weight: -0.0126
Feature 32: 8 with weight: -0.0124
Feature 33: 61 with weight: 0.0123
Feature 34: 70 with weight: 0.0121
Feature 35: 33 with weight: 0.0121
Feature 36: 73 with weight: 0.0118
Feature 37: 74 with weight: -0.0117
Feature 38: 6 with weight: -0.0113
Feature 39: 87 with weight: -0.0111
Feature 40: 12 with weight: -0.0109
Feature 41: 46 with weight: -0.0105
Feature 42: 7 with weight: 0.0098
Feature 43: 80 with weight: 0.0097
Feature 44: 50 with weight: 0.0096
Feature 45: 47 with weight: -0.0095
Feature 46: 68 with weight: 0.0090
Feature 47: 76 with weight: -0.0086
Feature 48: 18 with weight: 0.0081
Feature 49: 39 with weight: 0.0080
Feature 50: 20 with weight: 0.0078
Feature 51: 51 with weight: -0.0077
Feature 52: 17 with weight: 0.0071
Feature 53: 15 with weight: 0.0071
Feature 54: 16 with weight: -0.0070
Feature 55: 81 with weight: 0.0070
Feature 56: 55 with weight: -0.0069
Feature 57: 78 with weight: 0.0069
Feature 58: 38 with weight: -0.0067
Feature 59: 69 with weight: 0.0062
Feature 60: 72 with weight: -0.0062
Feature 61: 59 with weight: 0.0059
Feature 62: 34 with weight: 0.0058
Feature 63: 79 with weight: 0.0052
Feature 64: 35 with weight: 0.0048
Feature 65: 10 with weight: 0.0047
Feature 66: 0 with weight: -0.0047
Feature 67: 89 with weight: -0.0046
Feature 68: 3 with weight: -0.0042
Feature 69: 54 with weight: 0.0039
Feature 70: 25 with weight: -0.0037
Feature 71: 44 with weight: 0.0037
Feature 72: 63 with weight: 0.0036
Feature 73: 91 with weight: -0.0029
Feature 74: 13 with weight: -0.0029
Feature 75: 45 with weight: -0.0024
Feature 76: 32 with weight: 0.0023
Feature 77: 2 with weight: 0.0023
```

```
Feature 78: 85 with weight: 0.0023
Feature 79: 14 with weight: 0.0023
Feature 80: 49 with weight: 0.0022
Feature 81: 42 with weight: 0.0022
Feature 82: 57 with weight: -0.0022
Feature 83: 19 with weight: -0.0022
Feature 84: 5 with weight: 0.0020
Feature 85: 83 with weight: -0.0019
Feature 86: 1 with weight: 0.0014
Feature 87: 29 with weight: -0.0013
Feature 88: 48 with weight: 0.0006
Feature 89: 4 with weight: -0.0003
Feature 90: 60 with weight: -0.0002
Feature 91: 36 with weight: -0.0001
Feature 92: 22 with weight: 0.0001
Feature 93: 92 with weight: 0.0001
```



Using Saved Model

In [ ]:

```
def predict_and_display(image_path, model, class_labels):
    #Loading and preprocessing the image
    img = image.load_img(image_path, target_size=(224, 224))
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array = preprocess_input(img_array)

    #Making a prediction
    prediction = model.predict(img_array)
    predicted_class_index = np.argmax(prediction)

    #Getting the class name from the manually defined list of class labels
    predicted_class_label = class_labels[predicted_class_index]

    #Displaying the predicted class name
    plt.imshow(img)
```

```
    plt.axis('off')
    plt.title(f"Predicted Disease: {predicted_class_label}")
    plt.show()

#Loading trained model
model.load_weights('D:/Big Data Analytics/Term-2/BDM 3014 - Introduction to Artif
icial Intelligence 01/Final Project/Development/prediction_model_v2.weights.h5')

#Define your class labels (in order)
class_labels = ['Anthracnose', 'Bacterial Canker', 'Cutting Weevil', 'Die Back',
'Gall Midge', 'Healthy', 'Powdery Mildew', 'Sooty Mould']  # Replace with your ac
tual class names

#Image path to test the model
image_path_to_test = 'D:/Big Data Analytics/Term-2/BDM 3014 - Introduction to Art
ificial Intelligence 01/Final Project/20211231_123327 (Custom).jpg'
predict_and_display(image_path_to_test, model, class_labels)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 144ms/step
```

Predicted Disease: Healthy

UI with Streamlit

In [2]:

```
import streamlit as st
from PIL import Image
from tensorflow.keras.preprocessing import image as keras_image
from tensorflow.keras.applications.efficientnet import preprocess_input
import numpy as np
from tensorflow.keras.models import load_model
import firebase_admin
from firebase_admin import credentials, firestore

#Caching the model and Firebase initialization for faster operations
@st.cache_resource
def load_firebase():
    #Initialize Firebase only if it hasn't been initialized already to avoid dupl
icate initialization
    if not firebase_admin._apps:
        #Firebase credential from the directory
        cred = credentials.Certificate("disease-overview-firebase-adminsdk-8kos6-
fae78f2fdb.json")
```

```
        firebase_admin.initialize_app(cred)
    return firestore.client()

#Custom CSS for Streamlit app
st.markdown("""
    <style>
        .main-container {
            width: 90%;
            margin: auto;
            padding: 20px;
            background-color: #f0f2f6;
            border-radius: 10px;
            box-shadow: 0px 4px 12px rgba(0, 0, 0, 0.1);
        }
        .header {
            font-size: 36px;
            font-weight: bold;
            color: #1f77b4;
            text-align: center;
            margin-bottom: 20px;
        }
        .description, .how-it-works {
            color: #333;
            font-size: 18px;
            text-align: justify;
        }
        .image-container {
            text-align: center;
            margin-bottom: 20px;
        }
        .how-it-works-header {
            font-size: 24px;
            font-weight: bold;
            color: #333;
            margin-top: 20px;
            text-align: justify;
        }
        .content-container {
            display: flex;
            justify-content: space-around;
            align-items: center;
            flex-wrap: wrap;
        }
        .column {
            flex: 1;
            margin: 10px;
            max-width: 48%;
        }
        @media screen and (max-width: 1200px) {
            .column {
                max-width: 100%;
                margin-bottom: 20px;
            }
        }
    </style>
""", unsafe_allow_html=True)


#Caching the model for faster predictions
@st.cache_resource
def load_trained_model():
    model = load_model('model_v2.h5')
    return model

#Caching the labels
def load_class_labels():
    class_labels = ['Anthracnose', 'Bacterial Canker', 'Cutting Weevil', 'Die Bac
k', 'Gall Midge', 'Healthy', 'Powdery Mildew', 'Sooty Mould']
```

```
    return class_labels


#Function to fetch disease details from Firestore
def get_disease_details(disease_name, db):
    #Matching disease name with the document name in Firestore
    doc_ref = db.collection('diseases').document(disease_name)
    doc = doc_ref.get()
    #Condition to implement only if the document exists in Firestore
    if doc.exists:
        disease_data = doc.to_dict()
        name = disease_data.get('name', 'Unknown')
        desc = disease_data.get('desc', 'No description available.')
        symptoms = disease_data.get('symptoms', 'No symptoms available.')
        soln = disease_data.get('soln', 'No control solution available.')
        return name, desc, symptoms, soln
    else:
        return disease_name, "Overview not available.", "Symptoms not available."
, "Control solution not available."

#Getting prediction using the trained model
#This function takes 3 inputs: image, model, and class labels
def predict_and_display(img, model, class_labels):
    #Ensuring the image has 3 channels (RGB); if it doesn't, convert the image
    if img.mode != 'RGB':
        img = img.convert('RGB')

    #Resizing the image to the required size
    img = img.resize((224, 224))
    img_array = keras_image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array = preprocess_input(img_array)

    #Prediction with model
    prediction = model.predict(img_array)
    #Prediction index with model
    predicted_class_index = np.argmax(prediction)
    #Prediction class based on index
    predicted_class_label = class_labels[predicted_class_index]
    #Prediction confidence
    confidence_percentage = 100 * np.max(prediction)

    #Returning label and confidence percentage
    return predicted_class_label, confidence_percentage


def main():
    #Streamlit layout with centered header
    st.markdown("<div class='main-container'><div class='header'>FarmAI</div>", u
nsafe_allow_html=True)

    #How it works section for UI
    st.markdown("<div class='how-it-works-header'><strong>How it Works</strong></
div>", unsafe_allow_html=True)
    st.markdown("""
    <div class='how-it-works'>
        1. <strong>Upload an Image</strong>: Select an image of a mango leaf show
ing symptoms of a potential disease.<br>
        2. <strong>Disease Overview</strong>: Once the image is uploaded, the app
displays information about the detected disease, including a description and symp
toms.<br>
        3. <strong>Compare Symptoms</strong>: Compare the visible symptoms on the
leaf with those provided to determine if further action may be needed.
    </div>
    """, unsafe_allow_html=True)

    st.markdown("<div class='description'>Upload an image of a mango leaf to view
information about the detected disease.</div>", unsafe_allow_html=True)
```

```
    #File uploader to upload image with Streamlit
    uploaded_file = st.file_uploader("Choose an image...", type=["jpg", "jpeg", "png"])

    #If an image is uploaded
    if uploaded_file is not None:
        #Initializing Firebase and model
        db = load_firebase()
        model = load_trained_model()
        class_labels = load_class_labels()

        image = Image.open(uploaded_file)

        st.markdown("<div class='content-container'>", unsafe_allow_html=True)

        #Creating two responsive columns
        col1, col2 = st.columns(2, gap="large")

        with col1:
            st.markdown("<div class='column image-container'>", unsafe_allow_html=True)
            #Displaying the uploaded image and the predicted disease with confidence
            st.image(image, caption="Uploaded Leaf Image", use_column_width=True)


            #Performing disease prediction and display the results
            predicted_class_label, confidence_percentage = predict_and_display(image, model, class_labels)

            #Condition where the confidence is low, and can't predict the disease properly for the given image
            if confidence_percentage < 30:
                st.write("Couldn't identify the disease with sufficient confidence from this image. Please upload a clearer image.")
            #When confidence is above threshold
            else:
                st.write(f"Predicted Disease: **{predicted_class_label}**")
                st.write(f"Confidence: **{confidence_percentage:.2f}%**")
            st.markdown("</div>", unsafe_allow_html=True)

        #Column 2 when the confidence is above the set threshold
        if confidence_percentage >= 30:
            with col2:
                st.markdown("<div class='column'>", unsafe_allow_html=True)
                #Fetching disease details from Firestore
                disease_name, overview, symptoms, soln = get_disease_details(predicted_class_label, db)

                #Showing the Disease Name on UI
                st.markdown(f"<div class='description'><strong>Disease Name:</strong> {disease_name}</div><br>", unsafe_allow_html=True)
                #Overview
                st.markdown(f"<div class='description'><strong>Overview:</strong> {overview}</div><br>", unsafe_allow_html=True)
                #Symptoms
                st.markdown(f"<div class='description'><strong>Symptoms:</strong> {symptoms}</div><br>", unsafe_allow_html=True)
                #Control
                st.markdown(f"<div class='description'><strong>Control:</strong> {soln}</div>", unsafe_allow_html=True)
                st.markdown("</div>", unsafe_allow_html=True)

        st.markdown("</div>", unsafe_allow_html=True)

    #Closing main-container
    st.markdown("</div>", unsafe_allow_html=True)
```

```
#Running the main
if __name__ == "__main__":
    main()
```

```
2024-12-05 17:46:08.474
  Warning: to view this Streamlit app on a browser, run it with the following
  command:

    streamlit run C:\Users\sudee\AppData\Roaming\Python\Python312\site-packages\ip
ykernel_launcher.py [ARGUMENTS]
```

Deployment with Flask

In [ ]:

```
from flask import Flask, request, jsonify
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image as keras_image
from tensorflow.keras.applications.efficientnet import preprocess_input
from PIL import Image
import numpy as np
import io

app = Flask(__name__)

#Loading the trained model
model = load_model('model_v2.h5')

#Defining classes from the dataset
class_labels = ['Anthracnose', 'Bacterial Canker', 'Cutting Weevil', 'Die Back',
'Gall Midge', 'Healthy', 'Powdery Mildew', 'Sooty Mould']

#Define the prediction and display function
def predict_and_display(img, model, class_labels):
    #Resize the image to the required input shape
    img = img.resize((224, 224))

    #Preprocess the image
    #image to array using keras
    img_array = keras_image.img_to_array(img)
    #Expanding dimensions to match model inpu
    img_array = np.expand_dims(img_array, axis=0)
    #Preprocessing the image as required by EfficientNett
    img_array = preprocess_input(img_array)

    #Making a prediction
    prediction = model.predict(img_array)
    predicted_class_index = np.argmax(prediction)

    #Getting the class name from the defined list of class labels
    predicted_class_label = class_labels[predicted_class_index]

    #Getting the confidence percentage for the predicted class
    confidence_percentage = 100 * np.max(prediction)

    #Printing the class index for debugging
    print(f"Predicted Class Index: {predicted_class_index}")

    #Return prediction details
    #Converting the index to int for JSON
    return predicted_class_label, int(predicted_class_index), confidence_percenta
ge


#Defining the API route for image prediction
@app.route('/predict', methods=['POST'])
```

```python
def predict():
    #handling missing file from API request
    if 'image' not in request.files:
        return jsonify({"error": "No file part in the request"}), 400

    #Getting the file from the request
    file = request.files['image']
    if file.filename == '':
        return jsonify({"error": "No selected file"}), 400

    #Opening and processing the image
    try:
        img = Image.open(file.stream)
        predicted_class_label, predicted_class_index, confidence_percentage = pre
dict_and_display(img, model, class_labels)

        #Returning the prediction as JSON
        return jsonify({
            'predicted_class': predicted_class_label,
            'predicted_class_index': predicted_class_index,
            'confidence_percentage': confidence_percentage
        })
    #Raising an exception if the image cannot be processed, with error code
    except Exception as e:
        return jsonify({"error": f"Error processing the image: {str(e)}"}), 500

#Defiing home route for API endpoint
@app.route('/')
def home():
    return "Welcome to the Image Classification API! Use the /predict endpoint to
upload an image for prediction."

#running the api
if __name__ == '__main__':
    app.run(debug=True)
```