Step 1: Mount the google drive on Google Collab Notebook and import the data set

```
# Load the Drive helper and mount
from google.colab import drive
# This will prompt for authorization.
drive.mount ('/content/drive')
```

```
Mounted at /content/drive
```

After mounting the drive, locate the folder where the data is stored to use it in this collab notebook.

```
# After executing the cell above, Drive files will be present in the following directory
!ls "/content/drive/My Drive/plant_disease_prediction_project"
```

```
data  Model
```

```
!ls "/content/drive/My Drive/plant_disease_prediction_project/data"
```

```
Bacterial_Spot  Common_Rust  Early_Blight
```

Step 2: Import the required libraries

Import all the required libraries, as we are using CNN model, all the required layers, activations and optimizers libraries should be imported.

```
!pip install tensorflow==2.12.0
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.image import imread
import cv2
import random
import os
from os import listdir
from PIL import Image
import tensorflow as tf
from keras.preprocessing import image
from tensorflow.keras.utils import img_to_array, array_to_img
from keras.optimizers import Adam
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Activation, Flatten, Dropout, Dense
from sklearn.model_selection import train_test_split
from keras.models import model_from_json
from keras.utils import to_categorical
```

```
Requirement already satisfied: tensorflow==2.12.0 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: flatbuffers>=2.0 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: gast<=0.4.0,>=0.2.1 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: jax>=0.3.15 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: keras<2.13,>=2.12.0 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: numpy<1.24,>=1.22 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages 
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: tensorboard<2.13,>=2.12 in /usr/local/lib/python3.10/c
Requirement already satisfied: tensorflow-estimator<2.13,>=2.12.0 in /usr/local/lib/p
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.10/
Requirement already satisfied: wrapt<1.15,>=1.11.0 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: ml-dtypes>=0.2.0 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: scipy>=1.9 in /usr/local/lib/python3.10/dist-packages 
Requirement already satisfied: google-auth<3,>=1.6.3 in /usr/local/lib/python3.10/dis
Requirement already satisfied: google-auth-oauthlib<1.1,>=0.5 in /usr/local/lib/pythc
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/li
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/python3.10/di
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.10/dis
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.10/
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: pyasn1<0.6.0,>=0.4.6 in /usr/local/lib/python3.10/dist
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.10/dist-pack
2.12.0
```

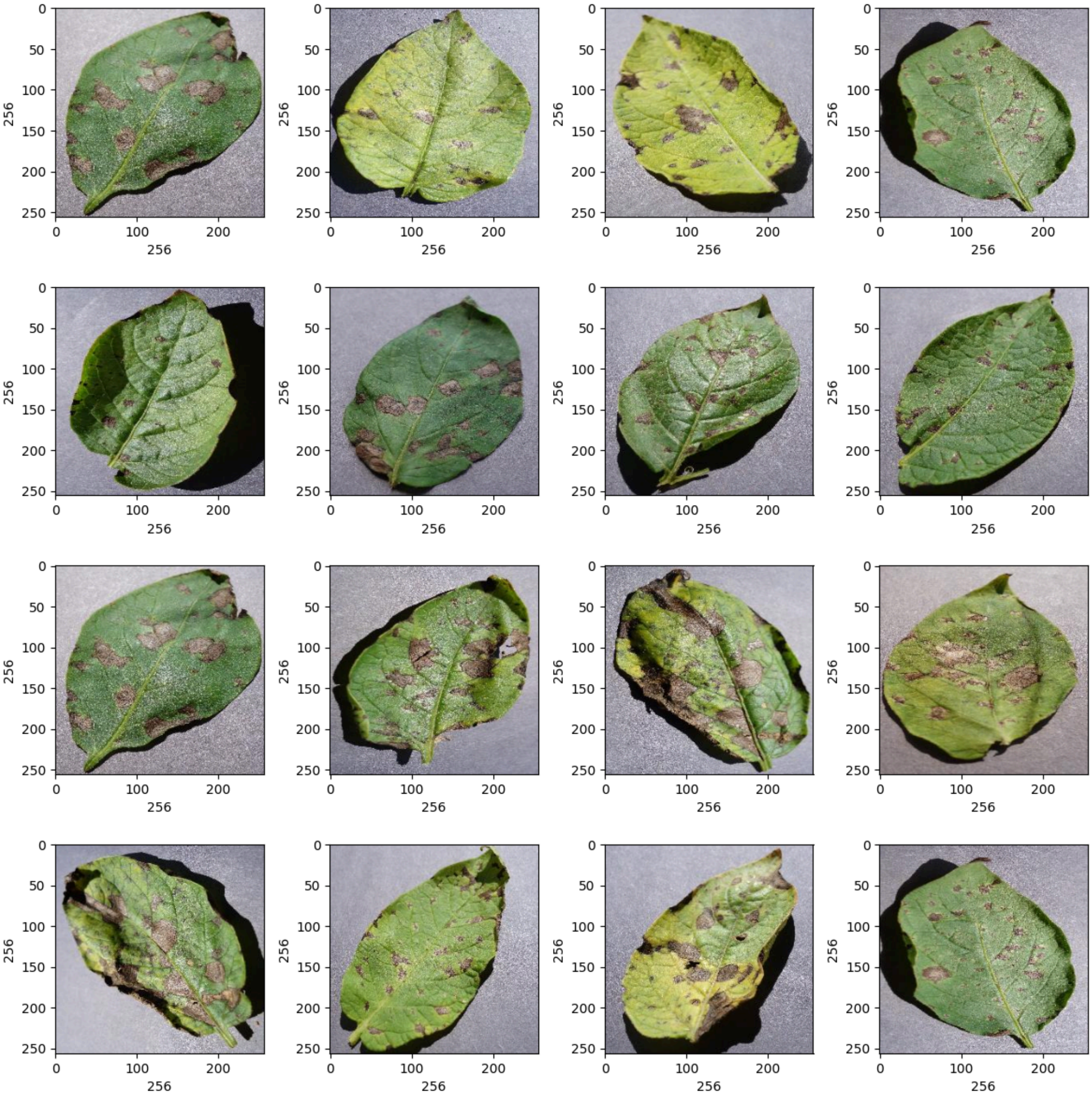
```
print(tf. __version__)
```

```
2.12.0
```

Step 3:Visualizing the images and resize images.

```
# Plotting 12 images to check dataset
plt. figure(figsize=(12,12))
path = "/content/drive/My Drive/plant_disease_prediction_project/data/Early_Blight"
for i in range(1,17):
```

```
    plt. subplot(4,4,i)
    plt.tight_layout ()
    rand_img = imread(path +'/'+ random.choice(sorted(os.listdir(path))))
    plt.imshow(rand_img)
    plt.xlabel(rand_img.shape[1], fontsize = 10)#width of image
    plt.ylabel(rand_img.shape[0], fontsize = 10)#height of image
```



Step 4: COnvert the images into a Numpy array and normalize them.

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

Now convert images into numpy array.

```
dir = "/content/drive/My Drive/plant_disease_prediction_project/data"
image_list, label_list = [], []
all_labels = ['Bacterial_Spot', 'Early_Blight', 'Common_Rust']
binary_labels = [0,1,2]
temp = -1
# Reading and converting image to numpy array
for directory in ['Bacterial_Spot', 'Early_Blight', 'Common_Rust']:
  plant_image_list = listdir(f"{dir}/{directory}")
  temp += 1
  for files in plant_image_list:
    image_path = f"{dir}/{directory}/{files}"
    image_list.append(convert_image_to_array(image_path))
    label_list.append(binary_labels[temp])
```

Step 5: Visualize the class count and check for class imbalance

```
# Visualize the number of classes count
label_counts = pd.DataFrame(label_list).value_counts()
label_counts.head()
```

```
0    300
1    300
2    300
dtype: int64
```

```
image_list[0].shape
```

```
(256, 256, 3)
```

Step 6: Splitting the dataset into train,validate and test sets.

```
x_train, x_test, y_train, y_test = train_test_split (image_list, label_list, test_size=0.
```

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

Step 7: Performing one-hot encoding on target variable.

```
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

Step 8: Creating the model architecture, compile the model and then fit it using the training data.

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

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

 max_pooling2d (MaxPooling2D  (None, 85, 85, 32)       0
 )

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

 max_pooling2d_1 (MaxPooling  (None, 42, 42, 16)       0
 2D)

 flatten (Flatten)           (None, 28224)             0

 dense (Dense)               (None, 8)                 225800

 dense_1 (Dense)             (None, 3)                 27

=================================================================
```

```
Total params: 231,347
Trainable params: 231,347
Non-trainable params: 0
```

Double-click (or enter) to edit

```
from sklearn.utils.class_weight import compute_sample_weight

# Assuming `y_train` is your training labels
sample_weights = compute_sample_weight('balanced', y_train)
```

```
model.compile(loss = 'categorical_crossentropy', optimizer = Adam(0.0001), metrics=['accu
#model.compile(optimizer= Adam(0.0001), loss='sparse_categorical_crossentropy', metrics=[
```

```
# Split the dataset without sample weights
x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_size=0.2, random

# Recalculate sample weights based on the training labels
sample_weights_train = compute_sample_weight('balanced', y_train)
sample_weights_val = compute_sample_weight('balanced', y_val)
```

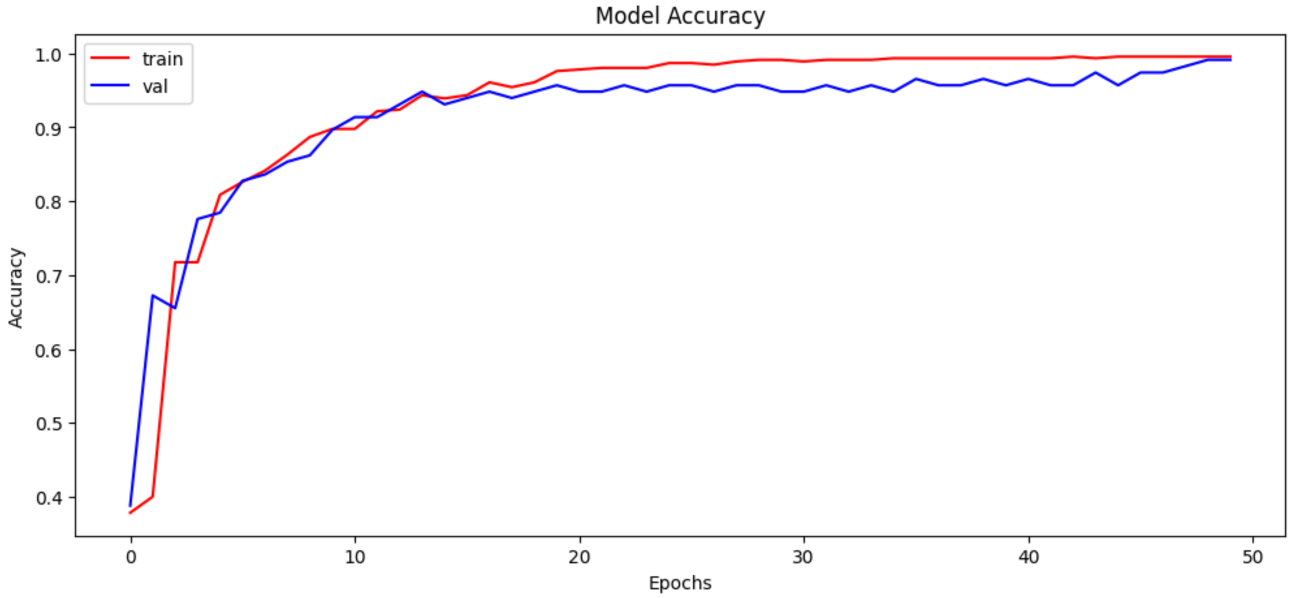
```
# Train the model
epochs = 50
batch_size = 128
history = model.fit(
    x_train, y_train,
    batch_size=batch_size,
    epochs=epochs,
    validation_data=(x_val, y_val),
    sample_weight=sample_weights_train
)
```

```
4/4 [==============================] - 30s 8s/step - loss: 0.0543 - accuracy: 0.98
Epoch 32/50
4/4 [==============================] - 31s 8s/step - loss: 0.0519 - accuracy: 0.99
Epoch 33/50
4/4 [==============================] - 34s 9s/step - loss: 0.0483 - accuracy: 0.99
Epoch 34/50
4/4 [==============================] - 32s 8s/step - loss: 0.0448 - accuracy: 0.99
Epoch 35/50
4/4 [==============================] - 29s 8s/step - loss: 0.0432 - accuracy: 0.99
Epoch 36/50
4/4 [==============================] - 39s 8s/step - loss: 0.0402 - accuracy: 0.99
Epoch 37/50
4/4 [==============================] - 30s 7s/step - loss: 0.0379 - accuracy: 0.99
Epoch 38/50
4/4 [==============================] - 30s 8s/step - loss: 0.0363 - accuracy: 0.99
Epoch 39/50
4/4 [==============================] - 30s 8s/step - loss: 0.0342 - accuracy: 0.99
Epoch 40/50
4/4 [==============================] - 32s 9s/step - loss: 0.0321 - accuracy: 0.99
Epoch 41/50
4/4 [==============================] - 28s 7s/step - loss: 0.0296 - accuracy: 0.99
Epoch 42/50
4/4 [==============================] - 30s 7s/step - loss: 0.0278 - accuracy: 0.99
Epoch 43/50
4/4 [==============================] - 30s 7s/step - loss: 0.0264 - accuracy: 0.99
Epoch 44/50
4/4 [==============================] - 30s 7s/step - loss: 0.0250 - accuracy: 0.99
Epoch 45/50
4/4 [==============================] - 31s 8s/step - loss: 0.0229 - accuracy: 0.99
Epoch 46/50
4/4 [==============================] - 30s 8s/step - loss: 0.0224 - accuracy: 0.99
Epoch 47/50
4/4 [==============================] - 32s 9s/step - loss: 0.0207 - accuracy: 0.99
Epoch 48/50
4/4 [==============================] - 30s 8s/step - loss: 0.0196 - accuracy: 0.99
Epoch 49/50
4/4 [==============================] - 30s 8s/step - loss: 0.0186 - accuracy: 0.99
Epoch 50/50
4/4 [==============================] - 30s 8s/step - loss: 0.0179 - accuracy: 0.99
```

```
model.save("/content/drive/My Drive/plant_disease_prediction_project/Model/plant_disease_
```

Step 9: Plot the accuracy and loss against each epoch

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




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

```
Calculating model accuracy
6/6 [==============================] - 5s 940ms/step - loss: 0.0803 - accuracy: 0.977
Test Accuracy: 97.77777791023254
```

Step 10:Make Predictions on testing data.

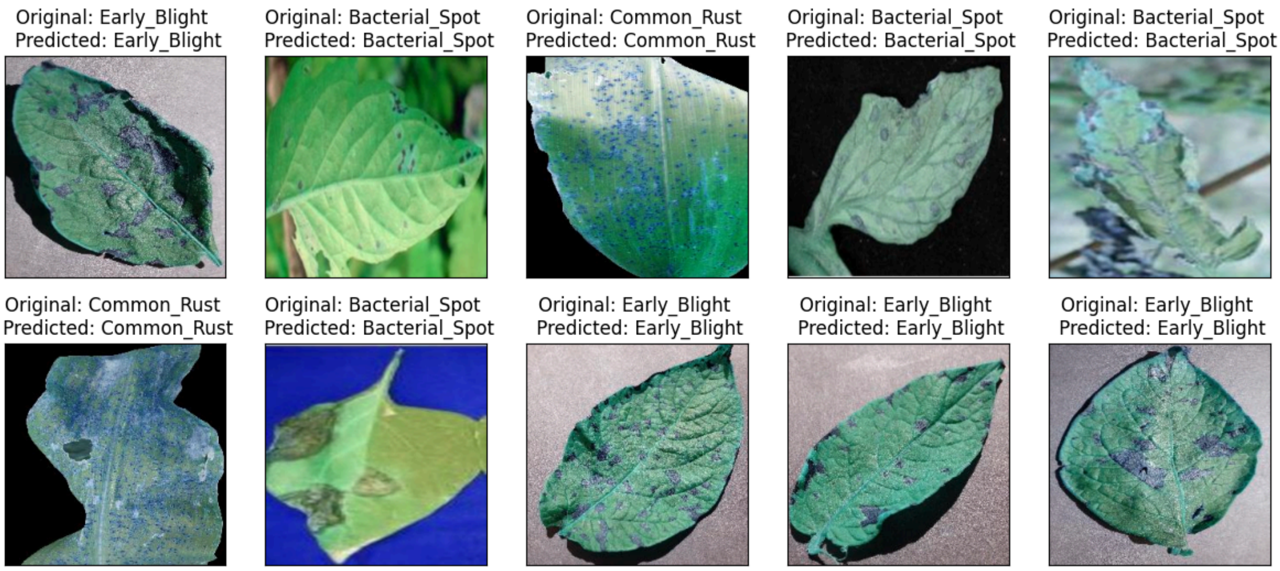
```
y_pred = model.predict(x_test)
```

```
6/6 [==============================] - 3s 461ms/step
```

```
# Visualizing the original and predicted labels for the test images
fig, axes = plt.subplots(2, 5, figsize=(15, 6), subplot_kw={'xticks':[], 'yticks':[]}, gr

for i, ax in enumerate(axes.flat):
    ax.imshow(array_to_img(x_test[i]))
    ax.set_title(f'Original: {all_labels[np.argmax(y_test[i])]} \n Predicted: {all_labels

plt.show()
```



Step 11: Visualizing the original and predicted labels for the test images.

```
#plotting image to compare
img = array_to_img(x_test[11])
img
```

```
# Finding max value from predition list and comparing original value vs predicted
print("Original Label: ",all_labels[np.argmax(y_test[11])])
print("Predicted Label: ",all_labels[np.argmax(y_pred[4])])
print(y_pred[2])
```

```
Original Label:  Bacterial_Spot
Predicted Label:  Bacterial_Spot
[7.6856650e-04 2.0632572e-03 9.9716812e-01]
```

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

```
Early_Blight - Early_Blight
Bacterial_Spot - Bacterial_Spot
Common_Rust - Common_Rust
Bacterial_Spot - Bacterial_Spot
Bacterial_Spot - Bacterial_Spot
Common_Rust - Common_Rust
Bacterial_Spot - Bacterial_Spot
Early_Blight - Early_Blight
Early_Blight - Early_Blight
Early_Blight - Early_Blight
Early_Blight - Early_Blight
Bacterial_Spot - Bacterial_Spot
Common_Rust - Common_Rust
Common_Rust - Common_Rust
Bacterial_Spot - Bacterial_Spot
Common_Rust - Common_Rust
Bacterial_Spot - Bacterial_Spot
Bacterial_Spot - Bacterial_Spot
Common_Rust - Early_Blight
Common_Rust - Common_Rust
Common_Rust - Common_Rust
Bacterial_Spot - Bacterial_Spot
Bacterial_Spot - Bacterial_Spot
Common_Rust - Common_Rust
Early_Blight - Early_Blight
Common_Rust - Common_Rust
Common_Rust - Common_Rust
Early_Blight - Early_Blight
Common_Rust - Common_Rust
Bacterial_Spot - Bacterial_Spot
Common_Rust - Common_Rust
Early_Blight - Early_Blight
Early_Blight - Early_Blight
Bacterial_Spot - Bacterial_Spot
Bacterial_Spot - Bacterial_Spot
Early_Blight - Early_Blight
Bacterial_Spot - Bacterial_Spot
Bacterial_Spot - Bacterial_Spot
Bacterial_Spot - Bacterial_Spot
Early_Blight - Early_Blight
Bacterial_Spot - Bacterial_Spot
Early_Blight - Early_Blight
Early_Blight - Early_Blight
Bacterial_Spot - Bacterial_Spot
Common_Rust - Common_Rust
Early_Blight - Early_Blight
```

```
Early_Blight - Early_Blight
Bacterial_Spot - Bacterial_Spot
Common_Rust - Common_Rust
Bacterial_Spot - Bacterial_Spot
```

```
import numpy as np
from tensorflow.keras.preprocessing import image  # Updated import statement
from keras.models import load_model
import matplotlib.pyplot as plt
from keras.applications.inception_v3 import preprocess_input

# Load the trained model
model = load_model("/content/drive/My Drive/plant_disease_prediction_project/Model/plant_

# Function to preprocess an image for prediction
def preprocess_image(image_path):
    img = image.load_img(image_path, target_size=(256, 256))
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array /= 255.0  # Normalize pixel values to the range [0, 1]
    return img_array

# Specify the path to a random image
random_image_path = "/content/drive/My Drive/plant_disease_prediction_project/data/Bacter

# Preprocess the image
preprocessed_image = preprocess_image(random_image_path)

# Make a prediction
predictions = model.predict(preprocessed_image)

# Decode the prediction to get the class label
class_label = np.argmax(predictions)

# Print the predicted class label
print("Predicted Class Label:", class_label)

# Map class label to class name (replace with your actual class names)
class_names = ['Bacterial_Spot', 'Early_Blight', 'Common_Rust']
predicted_class_name = class_names[class_label]

# Display the image
img = image.load_img(random_image_path, target_size=(256, 256))
plt.imshow(img)
plt.title(f'Predicted Class: {predicted_class_name}')
plt.show()
```

```
WARNING:tensorflow:5 out of the last 5 calls to <function Model.make_predict_functior
1/1 [==============================] - 0s 94ms/step
Predicted Class Label: 0
```


```
import numpy as np
from tensorflow.keras.preprocessing import image  # Updated import statement
from keras.models import load_model
import matplotlib.pyplot as plt
from keras.applications.inception_v3 import preprocess_input

# Load the trained model
model = load_model("/content/drive/My Drive/plant_disease_prediction_project/Model/plant_

# Function to preprocess an image for prediction
def preprocess_image(image_path):
    img = image.load_img(image_path, target_size=(256, 256))
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array /= 255.0  # Normalize pixel values to the range [0, 1]
    return img_array

# Specify the path to a random image
random_image_path = "/content/drive/My Drive/plant_disease_prediction_project/data/Early_

# Preprocess the image
preprocessed_image = preprocess_image(random_image_path)

# Make a prediction
predictions = model.predict(preprocessed_image)

# Decode the prediction to get the class label
class_label = np.argmax(predictions)

# Print the predicted class label
print("Predicted Class Label:", class_label)
```