Import the libaries as shown

```
import tensorflow as tf
from tensorflow.keras import models,layers
import matplotlib.pyplot as plt
```

```
# re-size  all the images to this
IMAGE_SIZE = [185, 500]
BATCH_SIZE = 32
CHANNELS = 3
EPOCHS  = 10
```

```
directory_path = ('/content/drive/MyDrive/data/data/train')

dataset = tf.keras.preprocessing.image_dataset_from_directory(
    directory_path,
    shuffle=True,
    image_size=(185, 500),
    batch_size=32
)
```

```
Found 4188 files belonging to 4 classes.
```

```
class_names = dataset.class_names
class_names
```

```
['Blight', 'Common_Rust', 'Gray_Leaf_Spot', 'Healthy']
```

```
len(dataset)
```

```
131
```

```
plt.figure(figsize=(10, 10))
for image_batch, labels_batch in dataset.take(1):
    for i in range(12):
        ax = plt.subplot(3, 4, i + 1)
        plt.imshow(image_batch[i].numpy().astype("uint8"))
        plt.title(class_names[labels_batch[i]])
        plt.axis("off")
```


```
len(dataset)
```

```
131
```

```
# 80% ==> training
# 20% ==>validation,10% test
```

```
train_size = 0.8
len(dataset)*train_size
```

```
104.80000000000001
```

```
train_ds = dataset.take(104)
len(train_ds)
```

```
104
```

```
test_ds = dataset.skip(104)
len(test_ds)
```

```
27
```

```
val_size = 0.1
len(dataset)*val_size
```

```
13.100000000000001
```

```
val_ds = test_ds.take(13)
len(val_ds)
```

```
13
```

```
test_ds = test_ds.skip(13)
len(test_ds)
```

```
14
```

```
def get_dataset_partition_tf(ds, train_split=0.8, val_split=0.1, test_split=0.1, shuffle=True, shuffle_size=10000):
    ds_size = len(ds)

    if shuffle:
        ds = ds.shuffle(shuffle_size, seed=12)
    train_size = int(train_split * ds_size)
    val_size = int(val_split * ds_size)

    train_ds = ds.take(train_size)

    val_ds = ds.skip(train_size).take(val_size)
    test_ds = ds.skip(train_size).skip(val_size)

    return train_ds, val_ds, test_ds
```

```
train_ds, val_ds, test_ds = get_dataset_partition_tf(dataset)
```

```
len(train_ds)
```

```
104
```

```
len(val_ds)
```

```
13
```

```
len(test_ds)
```

```
14
```

```
train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
val_ds = val_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
test_ds = test_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
```

```
from tensorflow.keras.layers import Resizing, Rescaling

resize_and_rescale = tf.keras.Sequential([
    Resizing(IMAGE_SIZE, IMAGE_SIZE),
    Rescaling(1./255),
])
```

```
data_augmentation = tf.keras.Sequential([
    layers.RandomFlip("horizontal_and_vertical"),
    layers.RandomRotation(0.2),
])
```

```
# Build the model
model = models.Sequential([
    layers.Input(shape=(IMAGE_SIZE[0], IMAGE_SIZE[1], CHANNELS)),
    data_augmentation,
    layers.Rescaling(1./255),
    layers.Conv2D(32, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(512, activation='relu'),
    layers.Dense(len(class_names), activation='softmax')
])

# Model summary
model.summary()
```

```
Model: "sequential_2"
```

Layer (type)	Output Shape	Param #
sequential_1 (Sequential)	(None, 185, 500, 3)	0
rescaling_1 (Rescaling)	(None, 185, 500, 3)	0
conv2d (Conv2D)	(None, 183, 498, 32)	896
max_pooling2d (MaxPooling2D)	(None, 91, 249, 32)	0
conv2d_1 (Conv2D)	(None, 89, 247, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 44, 123, 64)	0
conv2d_2 (Conv2D)	(None, 42, 121, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 21, 60, 128)	0
conv2d_3 (Conv2D)	(None, 19, 58, 128)	147,584
max_pooling2d_3 (MaxPooling2D)	(None, 9, 29, 128)	0
flatten (Flatten)	(None, 33408)	0
dense (Dense)	(None, 512)	17,105,408
dense_1 (Dense)	(None, 4)	2,052

```
Total params: 17,348,292 (66.18 MB)
Trainable params: 17,348,292 (66.18 MB)
Non-trainable params: 0 (0.00 B)
```

```
model.compile(
    optimizer='adam',
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    metrics=['accuracy']
)
```

```
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.optimizers import Adam

model.compile(optimizer=Adam(), loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Set up early stopping
early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=5,
    restore_best_weights=True
)

# Train the model
history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=10,
    callbacks=[early_stopping]
)
```

```
Epoch 1/10
104/104 ━━━━━━━━━━━━━━━━━━━━ 925s 9s/step - accuracy: 0.7890 - loss: 0.5288 - val_accuracy: 0.8606 - val_loss: 0.3
Epoch 2/10
104/104 ━━━━━━━━━━━━━━━━━━━━ 965s 9s/step - accuracy: 0.8191 - loss: 0.4317 - val_accuracy: 0.8558 - val_loss: 0.4
Epoch 3/10
104/104 ━━━━━━━━━━━━━━━━━━━━ 898s 8s/step - accuracy: 0.8214 - loss: 0.4421 - val_accuracy: 0.8582 - val_loss: 0.3
Epoch 4/10
104/104 ━━━━━━━━━━━━━━━━━━━━ 934s 9s/step - accuracy: 0.8149 - loss: 0.4282 - val_accuracy: 0.8726 - val_loss: 0.3
Epoch 5/10
104/104 ━━━━━━━━━━━━━━━━━━━━ 881s 8s/step - accuracy: 0.8531 - loss: 0.3505 - val_accuracy: 0.7260 - val_loss: 0.8
Epoch 6/10
104/104 ━━━━━━━━━━━━━━━━━━━━ 890s 9s/step - accuracy: 0.8529 - loss: 0.3597 - val_accuracy: 0.8750 - val_loss: 0.3
Epoch 7/10
104/104 ━━━━━━━━━━━━━━━━━━━━ 878s 8s/step - accuracy: 0.8583 - loss: 0.3321 - val_accuracy: 0.8870 - val_loss: 0.2
Epoch 8/10
104/104 ━━━━━━━━━━━━━━━━━━━━ 877s 8s/step - accuracy: 0.8822 - loss: 0.3119 - val_accuracy: 0.8750 - val_loss: 0.2
Epoch 9/10
104/104 ━━━━━━━━━━━━━━━━━━━━ 926s 8s/step - accuracy: 0.8573 - loss: 0.3596 - val_accuracy: 0.8942 - val_loss: 0.2
Epoch 10/10
104/104 ━━━━━━━━━━━━━━━━━━━━ 916s 8s/step - accuracy: 0.8663 - loss: 0.3386 - val_accuracy: 0.8894 - val_loss: 0.3
```

```
scores = model.evaluate(test_ds)
```

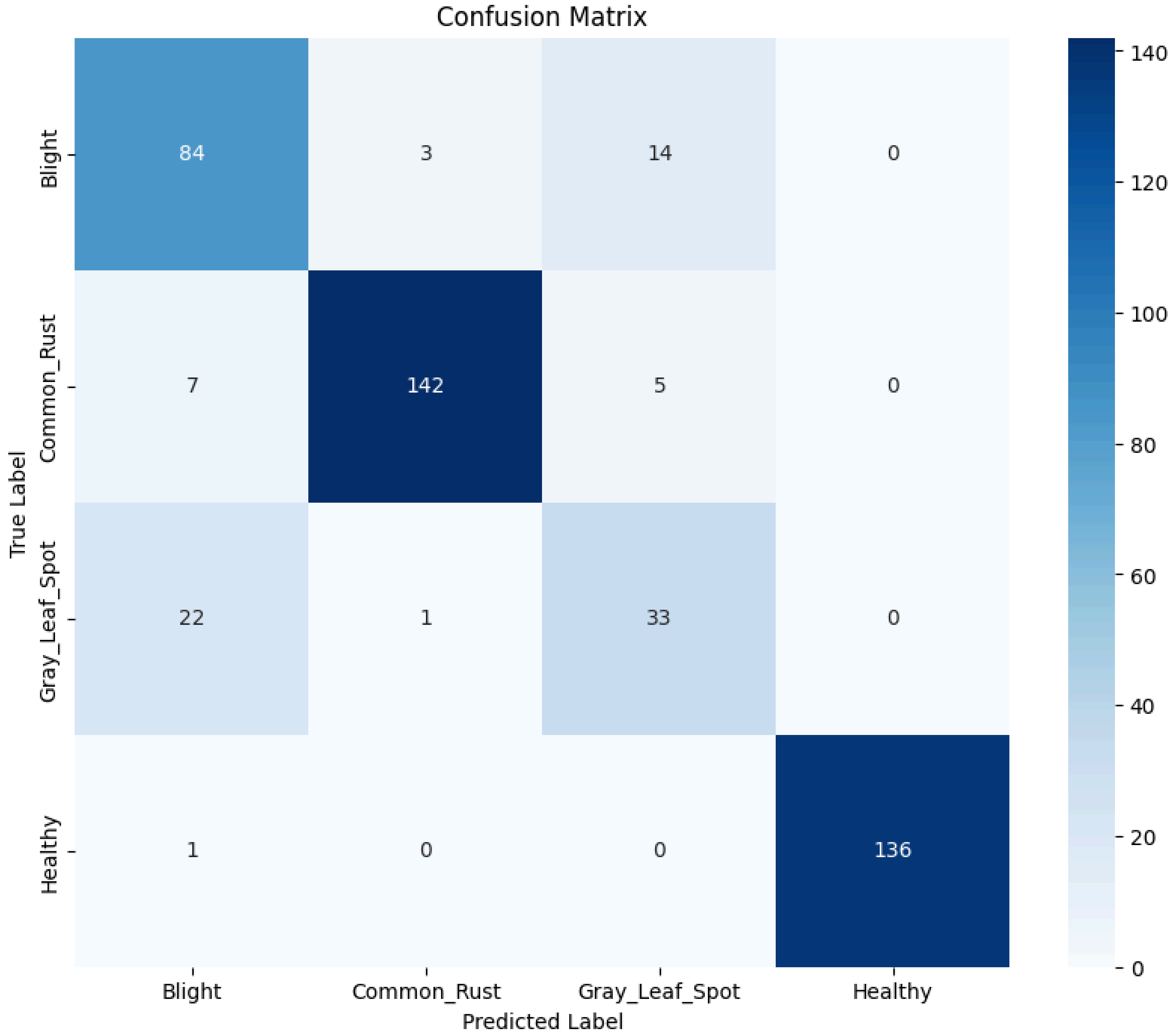
```
14/14 ━━━━━━━━━━━━━━━━━━━━ 60s 2s/step - accuracy: 0.8678 - loss: 0.2834
```

```
scores
```

```
[0.2650536596775055, 0.8816964030265808]
```

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import classification_report, confusion_matrix

# Function to get predictions and true labels
def get_predictions_and_labels(ds):
    images, labels = [], []
    for image_batch, label_batch in ds:
        images.extend(image_batch.numpy())
        labels.extend(label_batch.numpy())
    return np.array(images), np.array(labels)

# Get test images and labels
test_images, test_labels = get_predictions_and_labels(test_ds)

# Make predictions
predictions = model.predict(test_images)
predicted_labels = np.argmax(predictions, axis=1)

# Generate confusion matrix
cm = confusion_matrix(test_labels, predicted_labels)

# Plot confusion matrix
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()

# Generate and print classification report
report = classification_report(test_labels, predicted_labels, target_names=class_names)
print(report)
```

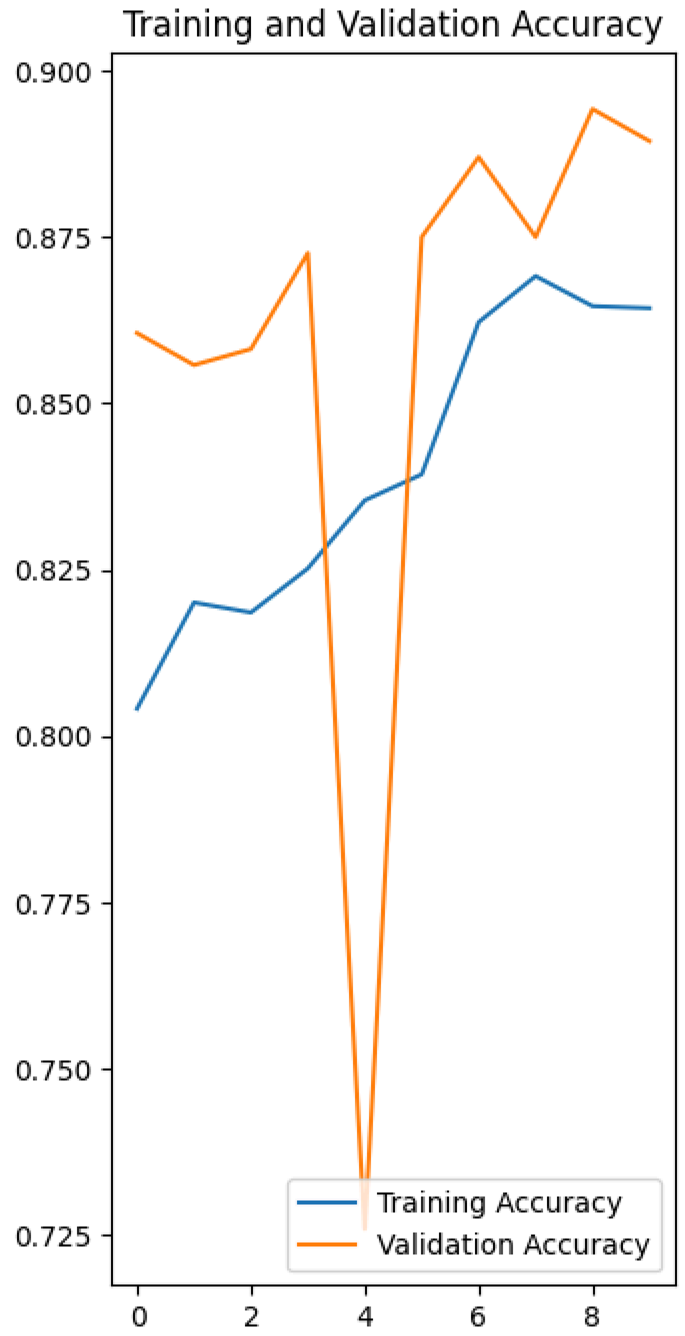
```
14/14 ━━━━━━━━━━━━━━━━━━━━ 33s 2s/step
```

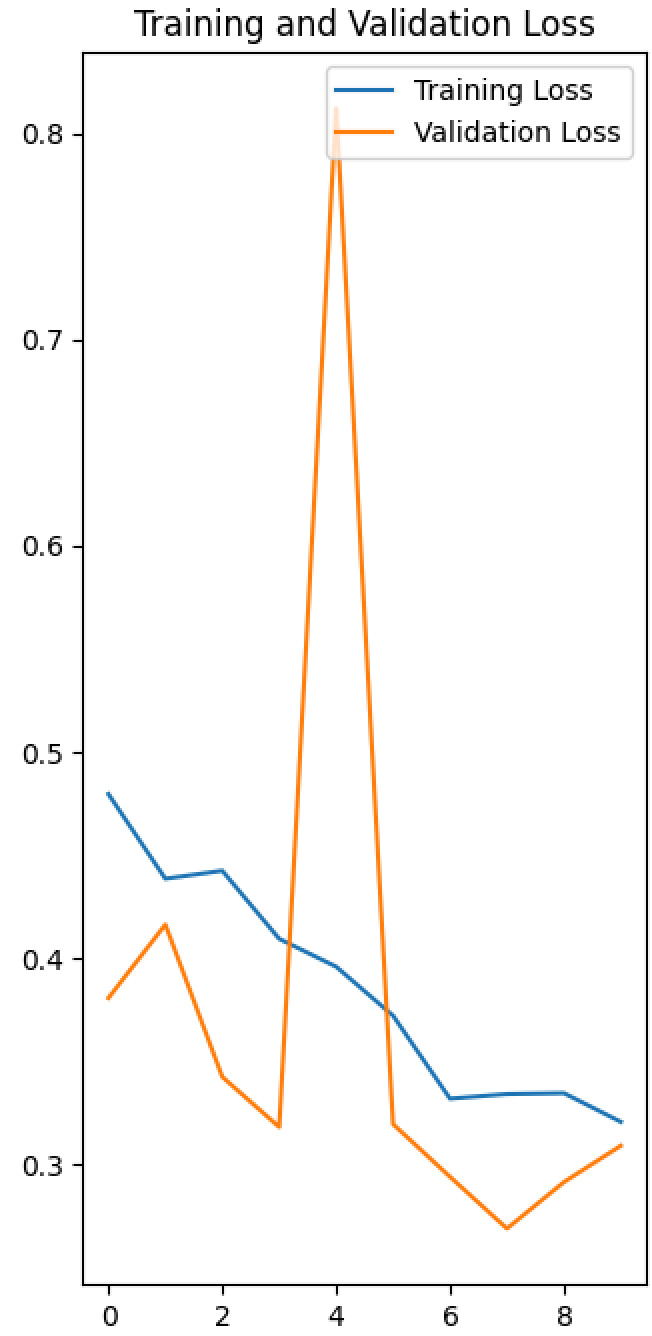



```
                precision    recall  f1-score   support

        Blight       0.74      0.83      0.78       101
   Common_Rust       0.97      0.92      0.95       154
Gray_Leaf_Spot       0.63      0.59      0.61        56
       Healthy       1.00      0.99      1.00       137

      accuracy                           0.88       448
     macro avg       0.84      0.83      0.83       448
  weighted avg       0.89      0.88      0.88       448
```

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']
```

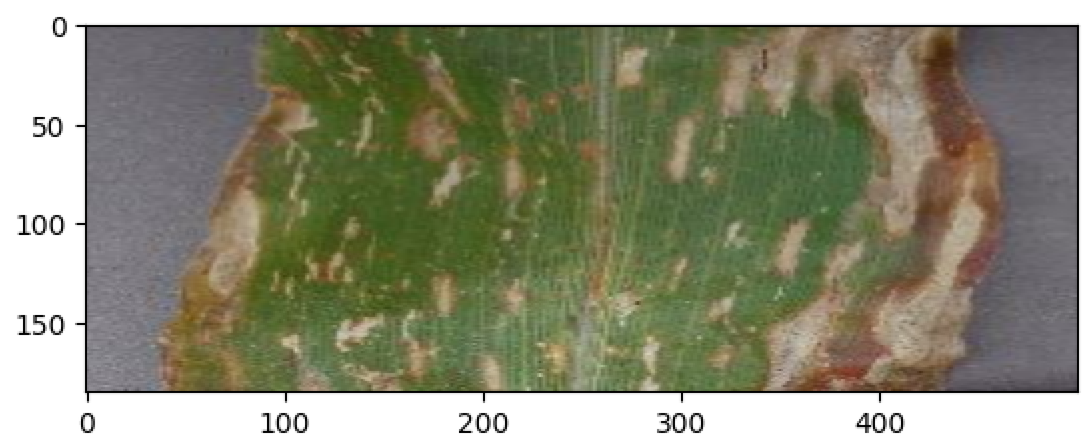
```
plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(range(10), acc, label='Training Accuracy')
plt.plot(range(10), val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')


plt.subplot(1, 2, 2)
plt.plot(range(10), loss, label='Training Loss')
plt.plot(range(10), val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```






```
import numpy as np
for images_batch, labels_batch in test_ds.take(1):
    first_image = images_batch[0].numpy().astype('uint8')
    first_label = labels_batch[0].numpy()

    print("first image to predict")
    plt.imshow(first_image)
    print("actual label:",class_names[first_label])

    batch_prediction = model.predict(images_batch)
    print("predicted label:",class_names[np.argmax(batch_prediction[0])])
```

```
first image to predict
actual label: Gray_Leaf_Spot
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
predicted label: Gray_Leaf_Spot
```




```
def predict(model, img):
    img_array = tf.keras.preprocessing.image.img_to_array(images[i].numpy())
    img_array = tf.expand_dims(img_array, 0)
    predictions = model.predict(img_array)
    predicted_class = class_names[np.argmax(predictions[0])]
    confidence = round(100 * (np.max(predictions[0])), 2)
    return predicted_class, confidence
```

```python
plt.figure(figsize=(15, 15))
for images, labels in test_ds.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))

        predicted_class, confidence = predict(model, images[i].numpy())
        actual_class = class_names[labels[i]]

        plt.title(f"Actual: {actual_class},\n Predicted: {predicted_class}.\n Confidence: {confidence}%")
        plt.axis("off")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 125ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 128ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 140ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 132ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 132ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 133ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 131ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 131ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 134ms/step
```