Data Loading

In [1]:
```
import tensorflow as tf
from tensorflow.keras import models, layers
import matplotlib.pyplot as plt
import numpy as np
import os
```

In [2]:
```
IMAGE_SIZE = 224
BATCH_SIZE = 32
CHANNELS = 3
EPOCHS =10
```

In [3]:
```
dataset = tf.keras.preprocessing.image_dataset_from_directory(
    'rice_leaf_disease_images',
    shuffle = True,
    image_size = (IMAGE_SIZE, IMAGE_SIZE),
    batch_size = BATCH_SIZE
)
```

```
Found 5933 files belonging to 4 classes.
```

In [4]:
```
class_names = dataset.class_names
class_names
```

Out[4]:
```
['Bacterialblight', 'Blast', 'Brownspot', 'Tungro']
```

In [5]:
```
len(dataset) #186*32=5931
```

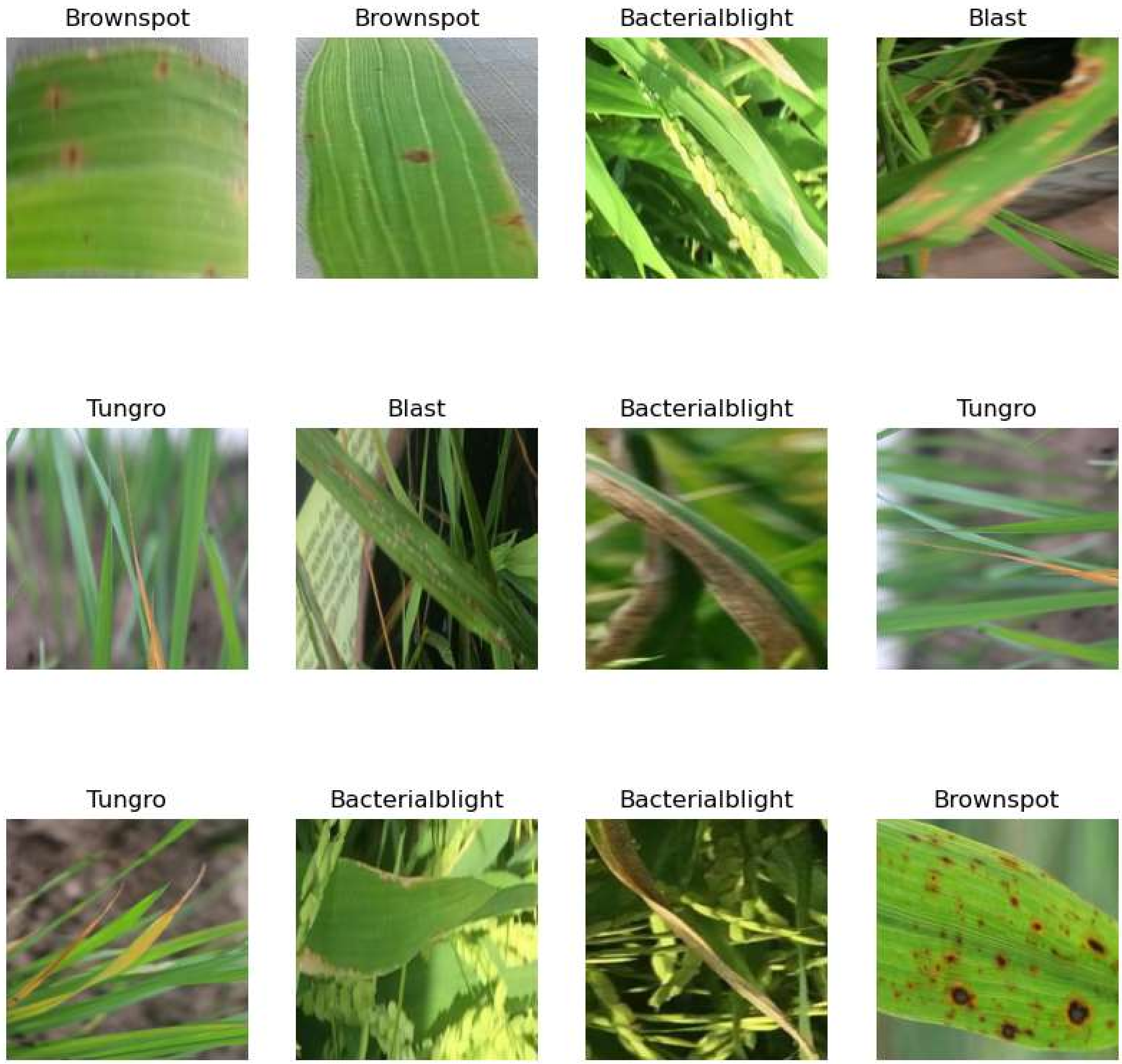
Out[5]:
```
186
```

In [6]:
```
# One random batch of images
for image_batch, label_batch in dataset.take(1):
    print(image_batch.shape)
    print(label_batch.numpy())
```

```
(32, 224, 224, 3)
[2 2 0 3 0 2 0 2 3 3 2 3 1 2 0 3 3 3 3 2 2 1 1 3 0 0 3 3 2 2 0 1]
```

In [7]:
```
plt.figure(figsize=(10,10))
for image_batch, label_batch in dataset.take(1):
    print(image_batch.shape)
    print(label_batch.numpy())
    for i in range(12): #showing 12 images out of 32
        ax = plt.subplot(3,4,i+1)
        plt.imshow(image_batch[i].numpy().astype("uint8"))
        plt.title(class_names[label_batch[i]])
        plt.axis("off")
```

```
(32, 224, 224, 3)
[2 2 0 1 3 1 0 3 3 0 0 2 1 2 0 2 2 2 0 3 1 1 3 3 0 0 1 0 3 3 2 2]
```



In [8]:
```
# (32=batch_size, 256, 256=image_size, 0 to 3=typesofdiseases)
# 0 - Bacterial Blight
# 1 - Blast
# 2 - Brownspot
# 3 - Tungro
```

In [9]:
```
# Spitting dataset for training, validation and testing
# 80% for training 10% for validation and 10% for testing
def get_dataset_partitions_tf(ds, train_split=0.8, val_split=0.1, test_split=0
    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
```

In [10]:
```
train_ds, val_ds, test_ds =get_dataset_partitions_tf(dataset)
```

In [11]:
```
# Catching and prefeching
train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUN
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)
```

Preprocessing

In [12]:
```
# Layer for resizing and rescaling
resize_and_rescale = tf.keras.Sequential([
    layers.experimental.preprocessing.Resizing(IMAGE_SIZE, IMAGE_SIZE),
    layers.experimental.preprocessing.Rescaling(1.0/255)
])
```

In [13]:
```
# Data Augmentation
data_augmentation = tf.keras.Sequential([
    layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical"),
    layers.experimental.preprocessing.RandomRotation(0.2)
])
```

VGG16

In [14]:
```
from tensorflow.keras.applications.resnet50 import ResNet50
```

In [15]:
```
resnet50 = ResNet50(input_shape=(IMAGE_SIZE,IMAGE_SIZE,CHANNELS),weights='imag
```

In [16]:
```
# Don't train existing weights
for layer in resnet50.layers:
    layer.trainable = False
```

In [17]:
```
x = tf.keras.layers.Flatten()(resnet50.output)
```

In [18]:
```
prediction = tf.keras.layers.Dense(len(class_names),activation='softmax')(x)
```

In [19]:
```
model = tf.keras.Model(inputs=resnet50.input, outputs=prediction)
```

In [20]:
```
model.summary()
```

```
 conv2_block1_out (Activation)  (None, 56, 56, 256)  0           ['conv2_b
lock1_add[0][0]']

 conv2_block2_1_conv (Conv2D)   (None, 56, 56, 64)   16448       ['conv2_b
lock1_out[0][0]']

 conv2_block2_1_bn (BatchNormal  (None, 56, 56, 64)  256         ['conv2_b
lock2_1_conv[0][0]']
 ization)

 conv2_block2_1_relu (Activatio  (None, 56, 56, 64)  0           ['conv2_b
lock2_1_bn[0][0]']
 n)

 conv2_block2_2_conv (Conv2D)   (None, 56, 56, 64)   36928       ['conv2_b
lock2_1_relu[0][0]']

 conv2_block2_2_bn (BatchNormal  (None, 56, 56, 64)  256         ['conv2_b
lock2_2_conv[0][0]']
 ization)
```

In [21]:
```
model.compile(
    optimizer='adam',
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    metrics=['accuracy'])
```

In [22]:
```
logdir='logs'
```

In [23]:
```
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=logdir)
```

In [24]:
```
callback = tf.keras.callbacks.EarlyStopping(
    monitor="val_loss",
    min_delta=0.001,
    patience=6,
    verbose=2,
    mode="auto",
    baseline=None,
    restore_best_weights=False,
)
```

In [25]:
```
history = model.fit(
    train_ds,
    epochs=EPOCHS,
    batch_size=BATCH_SIZE,
    verbose=2,
    validation_data=val_ds,
    callbacks=callback
)
```

```
Epoch 1/10
148/148 [==============================] - 457s 3s/step - loss: 1.2935 - accu
racy: 0.9201 - val_loss: 0.0288 - val_accuracy: 0.9931
Epoch 2/10
148/148 [==============================] - 387s 3s/step - loss: 0.0693 - accu
racy: 0.9913 - val_loss: 0.0817 - val_accuracy: 0.9861
Epoch 3/10
148/148 [==============================] - 404s 3s/step - loss: 0.0469 - accu
racy: 0.9928 - val_loss: 0.0483 - val_accuracy: 0.9965
Epoch 4/10
148/148 [==============================] - 391s 3s/step - loss: 0.0513 - accu
racy: 0.9945 - val_loss: 0.0260 - val_accuracy: 0.9948
Epoch 5/10
148/148 [==============================] - 386s 3s/step - loss: 0.0321 - accu
racy: 0.9958 - val_loss: 0.0098 - val_accuracy: 0.9965
Epoch 6/10
148/148 [==============================] - 386s 3s/step - loss: 0.1063 - accu
racy: 0.9915 - val_loss: 0.0365 - val_accuracy: 0.9948
Epoch 7/10
148/148 [==============================] - 363s 2s/step - loss: 0.1077 - accu
racy: 0.9922 - val_loss: 0.0883 - val_accuracy: 0.9931
Epoch 8/10
148/148 [==============================] - 458s 3s/step - loss: 0.0396 - accu
racy: 0.9966 - val_loss: 0.0026 - val_accuracy: 0.9983
Epoch 9/10
148/148 [==============================] - 471s 3s/step - loss: 0.0168 - accu
racy: 0.9977 - val_loss: 6.1550e-06 - val_accuracy: 1.0000
Epoch 10/10
148/148 [==============================] - 466s 3s/step - loss: 0.0047 - accu
racy: 0.9987 - val_loss: 4.0416e-04 - val_accuracy: 1.0000
```

In [26]:
```
scores = model.evaluate(test_ds)
scores
```

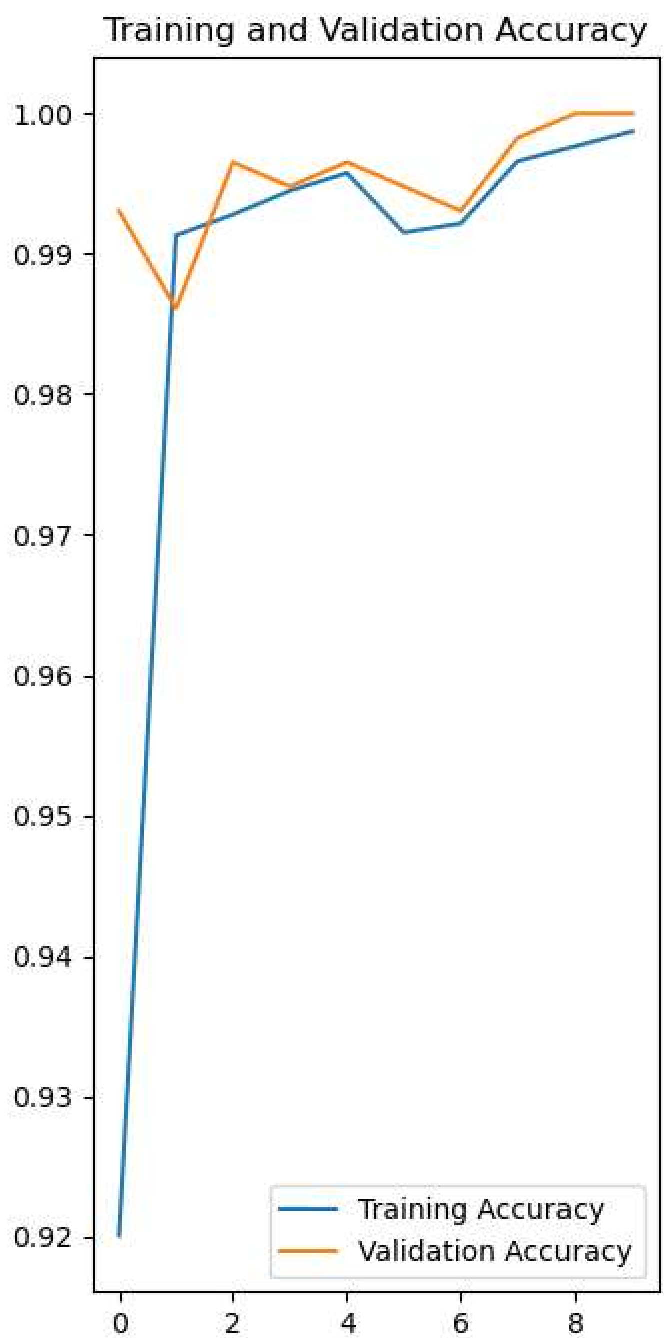
```
20/20 [==============================] - 84s 3s/step - loss: 0.0341 - accurac
y: 0.9984
```

Out[26]:
```
[0.03407653421163559, 0.9984375238418579]
```

In [27]:
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
```

In [28]:
```
plt.figure(figsize=(8,8))
plt.subplot(1,2,1)
plt.plot(range(EPOCHS), acc, label='Training Accuracy')
plt.plot(range(EPOCHS), val_acc, label='Validation Accuracy')
plt.legend(loc = 'lower right')
plt.title('Training and Validation Accuracy')
```

Out[28]:
```
Text(0.5, 1.0, 'Training and Validation Accuracy')
```



In [29]:
```
def predict(model, img):
    img_array = tf.keras.preprocessing.image.img_to_array(images[i].numpy())
    img_array = tf.expand_dims(img_array, 0)

    prediction = model.predict(img_array)

    prediction_class = class_names[np.argmax(prediction[0])]
    confidence = round(100 * (np.max(prediction[0])),2)

    return prediction_class, confidence
```

In [147]:
```
plt.figure(figsize=(15,15))
for images, labels in test_ds.take(4):
    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'predicted: {predicted_class},\n confidence: {confidence},
        plt.axis("off")
```

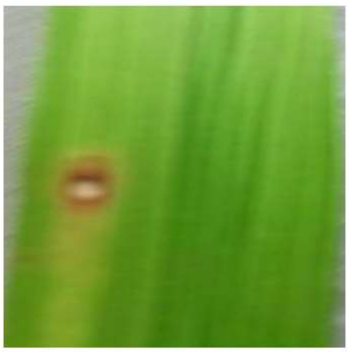
```
1/1 [==============================] - 0s 148ms/step
1/1 [==============================] - 0s 141ms/step
1/1 [==============================] - 0s 143ms/step
1/1 [==============================] - 0s 148ms/step
1/1 [==============================] - 0s 155ms/step
1/1 [==============================] - 0s 146ms/step
1/1 [==============================] - 0s 142ms/step
1/1 [==============================] - 0s 150ms/step
1/1 [==============================] - 0s 160ms/step
1/1 [==============================] - 0s 146ms/step
1/1 [==============================] - 0s 144ms/step
1/1 [==============================] - 0s 160ms/step
1/1 [==============================] - 0s 144ms/step
1/1 [==============================] - 0s 144ms/step
1/1 [==============================] - 0s 139ms/step
1/1 [==============================] - 0s 143ms/step
1/1 [==============================] - 0s 142ms/step
1/1 [==============================] - 0s 190ms/step
1/1 [==============================] - 0s 144ms/step
1/1 [==============================] - 0s 147ms/step
1/1 [==============================] - 0s 141ms/step
1/1 [==============================] - 0s 136ms/step
1/1 [==============================] - 0s 141ms/step
1/1 [==============================] - 0s 145ms/step
1/1 [==============================] - 0s 146ms/step
1/1 [==============================] - 0s 140ms/step
1/1 [==============================] - 0s 161ms/step
1/1 [==============================] - 0s 156ms/step
1/1 [==============================] - 0s 149ms/step
1/1 [==============================] - 0s 157ms/step
1/1 [==============================] - 0s 158ms/step
1/1 [==============================] - 0s 160ms/step
1/1 [==============================] - 0s 155ms/step
1/1 [==============================] - 0s 157ms/step
1/1 [==============================] - 0s 163ms/step
1/1 [==============================] - 0s 157ms/step
```

predicted: Bacterialblight,
confidence: 100.0,
Actual: Bacterialblight

predicted: Tungro,
confidence: 100.0,
Actual: Tungro

predicted: Bacterialblight,
confidence: 100.0,
Actual: Bacterialblight

predicted: Blast,
confidence: 100.0,
Actual: Blast

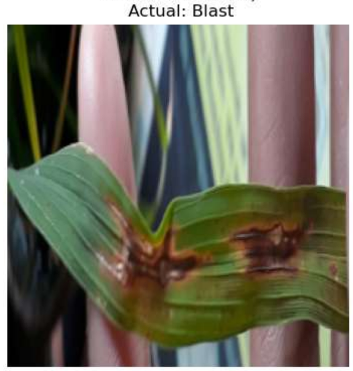


predicted: Brownspot,
confidence: 100.0,
Actual: Brownspot

predicted: Blast,
confidence: 100.0,
Actual: Blast

predicted: Bacterialblight,
confidence: 100.0,
Actual: Bacterialblight

predicted: Blast,
confidence: 100.0,
Actual: Blast



predicted: Blast,
confidence: 100.0,
Actual: Blast

In [ ]:

In [31]:

```
# Saving the model
model_version = max([int(i) for i in os.listdir("models") + [0]])+1
model.save(f'models\{model_version}')
```

```
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op, _jit_compiled_convolution_op, _jit_compiled_convolution_op, _jit_compiled_convolution_op, _jit_compiled_convolution_op while saving (showing 5 of 54). These functions will not be directly callable after loading.

INFO:tensorflow:Assets written to: models\17\assets

INFO:tensorflow:Assets written to: models\17\assets
```

In [158]:
```python
new_model = tf.keras.models.load_model('models/17')

# Check its architecture
new_model.summary()
```

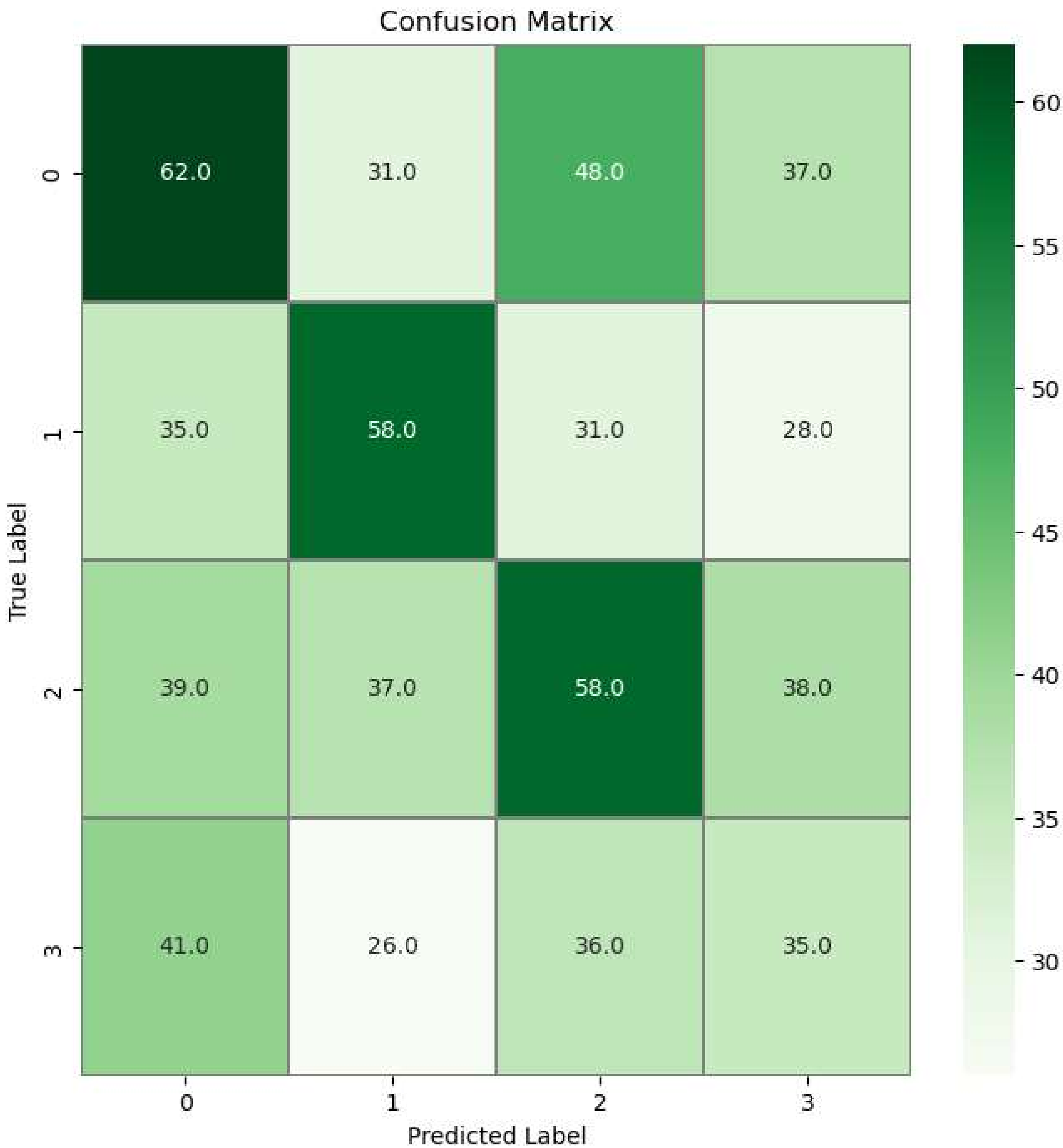
```
lock1_1_conv[0][0]']
 ization)

 conv2_block1_1_relu (Activatio  (None, 56, 56, 64)  0           ['conv2_b
lock1_1_bn[0][0]']
 n)

 conv2_block1_2_conv (Conv2D)   (None, 56, 56, 64)   36928       ['conv2_b
lock1_1_relu[0][0]']

 conv2_block1_2_bn (BatchNormal  (None, 56, 56, 64)  256         ['conv2_b
lock1_2_conv[0][0]']
 ization)

 conv2_block1_2_relu (Activatio  (None, 56, 56, 64)  0           ['conv2_b
lock1_2_bn[0][0]']
 n)

 conv2_block1_0_conv (Conv2D)   (None, 56, 56, 256)  16640       ['pool1_p
```

In [ ]:

In [33]:
```python
from sklearn.metrics import confusion_matrix , classification_report
```

In [159]:
```
# confusion matrix
import seaborn as sns
# Predict the values from the validation dataset
Y_pred = new_model.predict(test_ds)
# Convert predictions classes to one hot vectors
Y_pred_classes = np.argmax(Y_pred,axis = 1)
# Convert validation observations to one hot vectors
Y_true = tf.concat([y for x, y in test_ds], axis=0)
# compute the confusion matrix
confusion_mtx = confusion_matrix(Y_true, Y_pred_classes)
# plot the confusion matrix
f,ax = plt.subplots(figsize=(8, 8))
sns.heatmap(confusion_mtx, annot=True, linewidths=0.01,cmap="Greens",linecolor
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()
```

```
20/20 [==============================] - 42s 2s/step
```



In [160]:
```
print(classification_report(Y_true, Y_pred_classes, target_names=class_names))
```

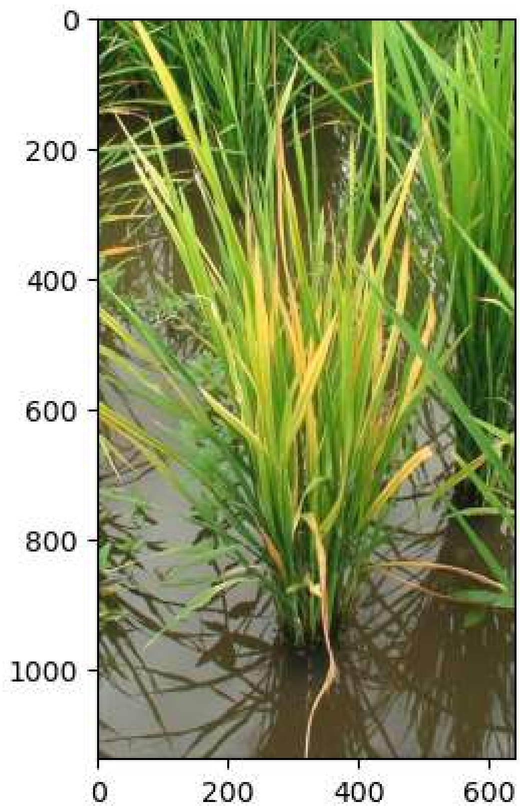
```
                precision    recall  f1-score   support

Bacterialblight       0.35      0.35      0.35       178
          Blast       0.38      0.38      0.38       152
      Brownspot       0.34      0.34      0.34       172
         Tungro       0.25      0.25      0.25       138

       accuracy                           0.33       640
      macro avg       0.33      0.33      0.33       640
   weighted avg       0.33      0.33      0.33       640
```

In [ ]:

In [36]:
```
import numpy as np
import cv2
```

In [165]:
```
img_path = cv2.imread(os.path.join('Testing Images','t.jfif'))
img_path = cv2.cvtColor(img_path,cv2.COLOR_BGR2RGB)
plt.imshow(img_path)
```

Out[165]:
```
<matplotlib.image.AxesImage at 0x26c467aa820>
```



In [166]:
```
img = cv2.resize(img_path,(224,224))
img = np.reshape(img,[1,224,224,3])
```

In [167]:
```
pred = new_model.predict(img)
```

```
1/1 [==============================] - 0s 146ms/step
```

In [168]:
```
pred
```

Out[168]:
```
array([[6.513488e-29, 0.000000e+00, 0.000000e+00, 1.000000e+00]],
      dtype=float32)
```

In [169]:
```
prediction_class = class_names[np.argmax(pred)]
prediction_class
```

Out[169]:
```
'Tungro'
```

In [ ]:

In [ ]:

In [ ]:

2 - vgg16 2epochs earlystopping 7 - vgg16 8 - vgg16 data + bb.jpg added

In [ ]:

In [ ]:

In [ ]: