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 =30
```

```
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 7926 files belonging to 5 classes.
```

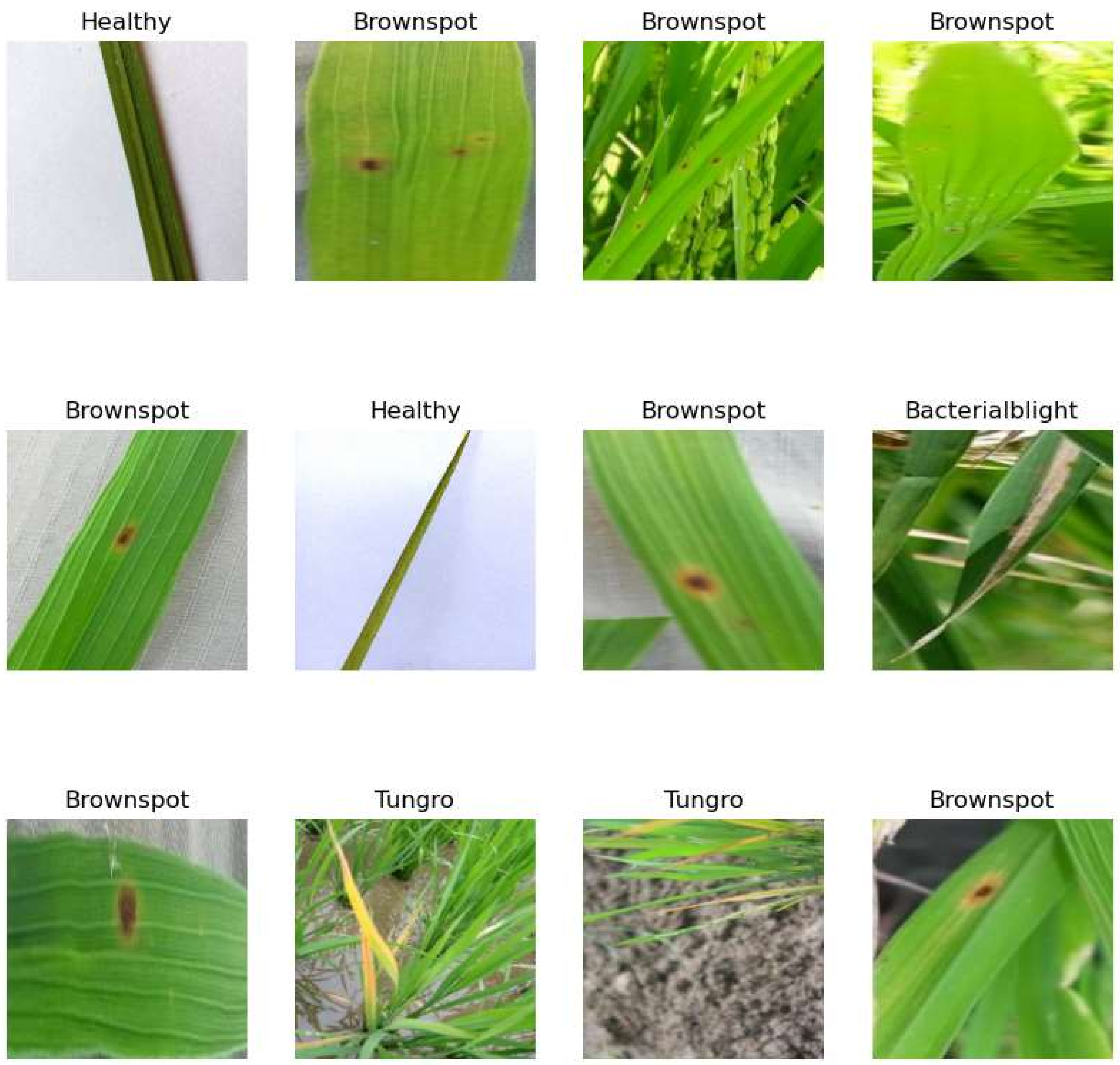
```
In [4]: class_names = dataset.class_names
        class_names

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

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

Out[5]: 248
```

```
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)
        [1 3 4 1 2 3 3 4 0 0 3 2 4 0 0 0 0 1 2 1 1 1 1 0 2 0 0 0 2 2 3 2]
```

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)
[3 2 2 2 2 3 2 0 2 4 4 2 1 4 2 2 0 1 2 2 0 2 1 4 1 3 4 2 0 2 4 1]
```



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().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.vgg16 import VGG16
```

In [15]:
```
vgg16 = VGG16(input_shape=(IMAGE_SIZE,IMAGE_SIZE,CHANNELS),weights='imagenet',
```

In [16]:
```
# Don't train existing weights
for layer in vgg16.layers:
    layer.trainable = False
```

In [17]:
```
x = tf.keras.layers.Flatten()(vgg16.output)
```

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

In [19]:
```
model = tf.keras.Model(inputs=vgg16.input, outputs=prediction)
```

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

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

 flatten (Flatten)           (None, 25088)             0

 dense (Dense)               (None, 5)                 125445

=================================================================
Total params: 14,840,133
Trainable params: 125,445
Non-trainable params: 14,714,688
_________________________________________________________________
```

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]:
```
history = model.fit(
    train_ds,
    epochs=EPOCHS,
    batch_size=BATCH_SIZE,
    verbose=1,
    validation_data=val_ds,
    callbacks=[tensorboard_callback]
)
```

```
Epoch 1/30
198/198 [==============================] - 1297s 6s/step - loss: 2.1488 - acc
uracy: 0.8927 - val_loss: 0.9133 - val_accuracy: 0.9531
Epoch 2/30
198/198 [==============================] - 1165s 6s/step - loss: 0.5011 - acc
uracy: 0.9725 - val_loss: 0.6232 - val_accuracy: 0.9674
Epoch 3/30
198/198 [==============================] - 1165s 6s/step - loss: 0.2833 - acc
uracy: 0.9831 - val_loss: 0.5731 - val_accuracy: 0.9740
Epoch 4/30
198/198 [==============================] - 1537s 8s/step - loss: 0.2353 - acc
uracy: 0.9845 - val_loss: 0.5233 - val_accuracy: 0.9740
Epoch 5/30
198/198 [==============================] - 1835s 9s/step - loss: 0.1392 - acc
uracy: 0.9908 - val_loss: 0.2885 - val_accuracy: 0.9818
Epoch 6/30
198/198 [==============================] - 1708s 9s/step - loss: 0.1105 - acc
uracy: 0.9929 - val_loss: 0.3135 - val_accuracy: 0.9844
Epoch 7/30
198/198 [==============================] - 1540s 8s/step - loss: 0.0832 - acc
uracy: 0.9949 - val_loss: 0.1527 - val_accuracy: 0.9961
Epoch 8/30
198/198 [==============================] - 1485s 8s/step - loss: 0.0733 - acc
uracy: 0.9945 - val_loss: 0.4575 - val_accuracy: 0.9896
Epoch 9/30
198/198 [==============================] - 1524s 8s/step - loss: 0.1739 - acc
uracy: 0.9910 - val_loss: 0.8641 - val_accuracy: 0.9766
Epoch 10/30
198/198 [==============================] - 1855s 9s/step - loss: 0.2123 - acc
uracy: 0.9910 - val_loss: 0.2036 - val_accuracy: 0.9870
Epoch 11/30
198/198 [==============================] - 1853s 9s/step - loss: 0.1116 - acc
uracy: 0.9954 - val_loss: 0.2666 - val_accuracy: 0.9870
Epoch 12/30
198/198 [==============================] - 1598s 8s/step - loss: 0.0638 - acc
uracy: 0.9965 - val_loss: 0.0793 - val_accuracy: 0.9961
Epoch 13/30
198/198 [==============================] - 1499s 8s/step - loss: 0.0664 - acc
uracy: 0.9970 - val_loss: 0.3193 - val_accuracy: 0.9870
Epoch 14/30
198/198 [==============================] - 1472s 7s/step - loss: 0.1434 - acc
uracy: 0.9938 - val_loss: 0.3297 - val_accuracy: 0.9922
Epoch 15/30
198/198 [==============================] - 1641s 8s/step - loss: 0.1278 - acc
uracy: 0.9940 - val_loss: 0.3938 - val_accuracy: 0.9844
Epoch 16/30
198/198 [==============================] - 1681s 8s/step - loss: 0.1026 - acc
uracy: 0.9967 - val_loss: 0.4645 - val_accuracy: 0.9896
Epoch 17/30
198/198 [==============================] - 1873s 9s/step - loss: 0.1097 - acc
uracy: 0.9965 - val_loss: 0.4900 - val_accuracy: 0.9883
Epoch 18/30
198/198 [==============================] - 2206s 11s/step - loss: 0.1961 - ac
curacy: 0.9930 - val_loss: 0.5340 - val_accuracy: 0.9857
Epoch 19/30
198/198 [==============================] - 2054s 10s/step - loss: 0.1214 - ac
curacy: 0.9957 - val_loss: 0.5983 - val_accuracy: 0.9883
```

```
Epoch 20/30
198/198 [==============================] - 1938s 10s/step - loss: 0.0378 - ac
curacy: 0.9983 - val_loss: 0.2611 - val_accuracy: 0.9935
Epoch 21/30
198/198 [==============================] - 1825s 9s/step - loss: 0.0396 - acc
uracy: 0.9976 - val_loss: 0.5703 - val_accuracy: 0.9909
Epoch 22/30
198/198 [==============================] - 1985s 10s/step - loss: 0.0782 - ac
curacy: 0.9972 - val_loss: 0.1955 - val_accuracy: 0.9935
Epoch 23/30
198/198 [==============================] - 2211s 11s/step - loss: 0.1338 - ac
curacy: 0.9949 - val_loss: 0.2378 - val_accuracy: 0.9948
Epoch 24/30
198/198 [==============================] - 2061s 10s/step - loss: 0.0613 - ac
curacy: 0.9968 - val_loss: 0.5224 - val_accuracy: 0.9909
Epoch 25/30
198/198 [==============================] - 2059s 10s/step - loss: 0.0330 - ac
curacy: 0.9984 - val_loss: 0.4245 - val_accuracy: 0.9922
Epoch 26/30
198/198 [==============================] - 2036s 10s/step - loss: 0.0119 - ac
curacy: 0.9992 - val_loss: 0.3353 - val_accuracy: 0.9935
Epoch 27/30
198/198 [==============================] - 1834s 9s/step - loss: 0.0674 - acc
uracy: 0.9979 - val_loss: 0.3629 - val_accuracy: 0.9935
Epoch 28/30
198/198 [==============================] - 2236s 11s/step - loss: 0.2007 - ac
curacy: 0.9959 - val_loss: 0.1345 - val_accuracy: 0.9922
Epoch 29/30
198/198 [==============================] - 1763s 9s/step - loss: 0.0714 - acc
uracy: 0.9979 - val_loss: 0.7482 - val_accuracy: 0.9857
Epoch 30/30
198/198 [==============================] - 1819s 9s/step - loss: 0.0484 - acc
uracy: 0.9978 - val_loss: 0.4536 - val_accuracy: 0.9935
```

In [25]:
```
scores = model.evaluate(test_ds)
scores
```

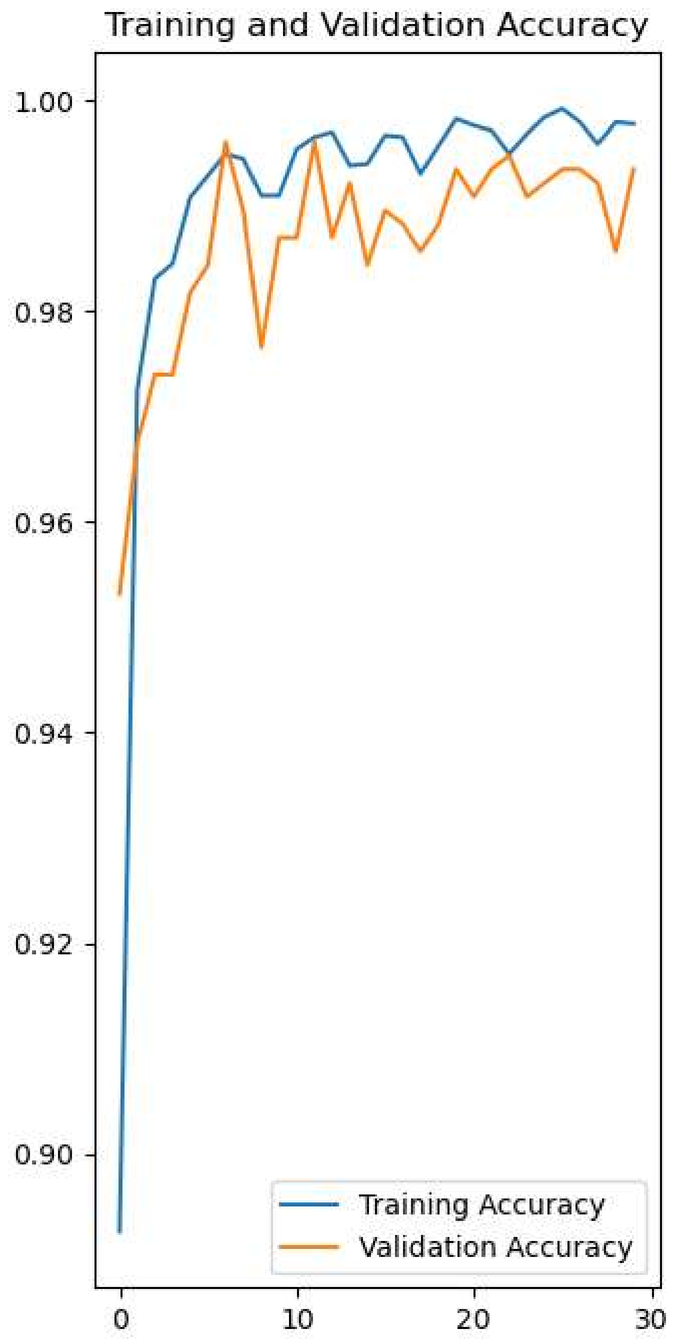
```
26/26 [==============================] - 304s 7s/step - loss: 0.4968 - accura
cy: 0.9940
```

Out[25]:
```
[0.49677610397338867, 0.9939903616905212]
```

In [26]:
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
```

In [27]:
```
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[27]:
```
Text(0.5, 1.0, 'Training and Validation Accuracy')
```



In [28]:
```
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 [29]:
```
plt.figure(figsize=(15,15))
for images, labels in test_ds.take(2):
    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 [==============================] - 1s 1s/step
1/1 [==============================] - 0s 259ms/step
1/1 [==============================] - 0s 221ms/step
1/1 [==============================] - 0s 309ms/step
1/1 [==============================] - 0s 263ms/step
1/1 [==============================] - 0s 251ms/step
1/1 [==============================] - 0s 253ms/step
1/1 [==============================] - 0s 338ms/step
1/1 [==============================] - 0s 254ms/step
1/1 [==============================] - 0s 310ms/step
1/1 [==============================] - 0s 242ms/step
1/1 [==============================] - 0s 267ms/step
1/1 [==============================] - 0s 281ms/step
1/1 [==============================] - 0s 264ms/step
1/1 [==============================] - 0s 283ms/step
1/1 [==============================] - 0s 254ms/step
1/1 [==============================] - 0s 278ms/step
1/1 [==============================] - 0s 251ms/step
```

In [33]:
```
# Saving the model
model_version = max([int(i) for i in os.listdir("new_models") + [0]])+1
model.save(f'new_models\{model_version}')
```

```
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op, _
jit_compiled_convolution_op, _jit_compiled_convolution_op, _jit_compiled_conv
olution_op, _jit_compiled_convolution_op while saving (showing 5 of 14). Thes
e functions will not be directly callable after loading.

INFO:tensorflow:Assets written to: new_models\6\assets

INFO:tensorflow:Assets written to: new_models\6\assets
```

In [ ]:

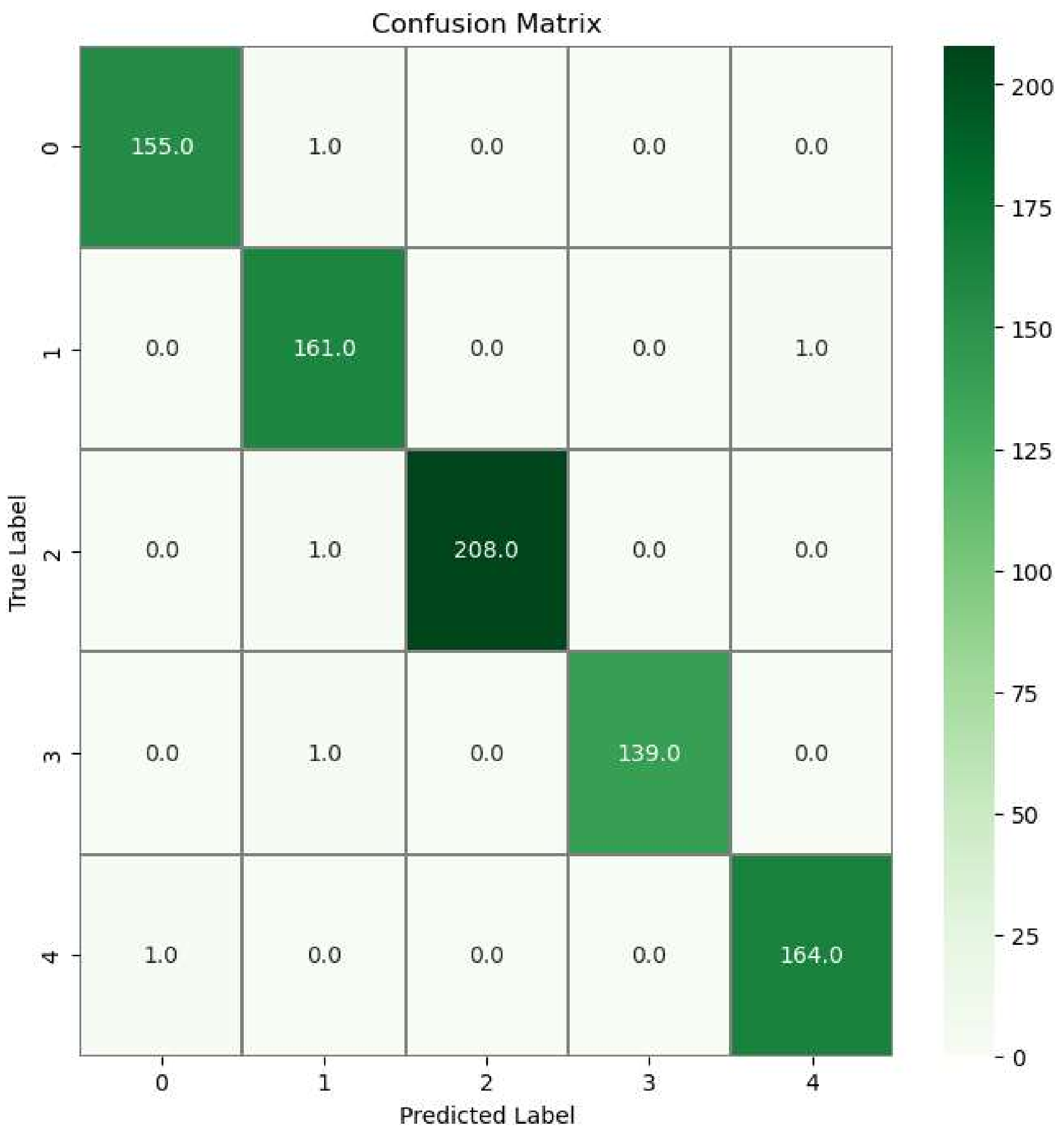
In [34]:
```
new_model = tf.keras.models.load_model('new_models/6')

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

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0         
                                                                 
 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792      
                                                                 
 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928     
                                                                 
 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0         
                                                                 
 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856     
                                                                 
 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584    
                                                                 
 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0         
                                                                 
 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168    
                                                                 
 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080    
                                                                 
 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080    
                                                                 
 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0         
                                                                 
 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160   
                                                                 
 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808   
                                                                 
 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808   
                                                                 
 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0         
                                                                 
 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808   
                                                                 
 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808   
                                                                 
 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808   
                                                                 
 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0         
                                                                 
 flatten (Flatten)           (None, 25088)             0         
                                                                 
 dense (Dense)               (None, 5)                 125445    
                                                                 
=================================================================
Total params: 14,840,133
Trainable params: 125,445
Non-trainable params: 14,714,688
_________________________________________________________________
```

In [35]:
```
from sklearn.metrics import confusion_matrix , classification_report
```

In [36]:
```
# 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()
```

```
26/26 [==============================] - 207s 8s/step
```



```
In [37]: print(classification_report(Y_true, Y_pred_classes, target_names=class_names))
```

```
                 precision    recall  f1-score   support

Bacterialblight       0.99      0.99      0.99       156
          Blast       0.98      0.99      0.99       162
      Brownspot       1.00      1.00      1.00       209
        Healthy       1.00      0.99      1.00       140
         Tungro       0.99      0.99      0.99       165

       accuracy                           0.99       832
      macro avg       0.99      0.99      0.99       832
   weighted avg       0.99      0.99      0.99       832
```

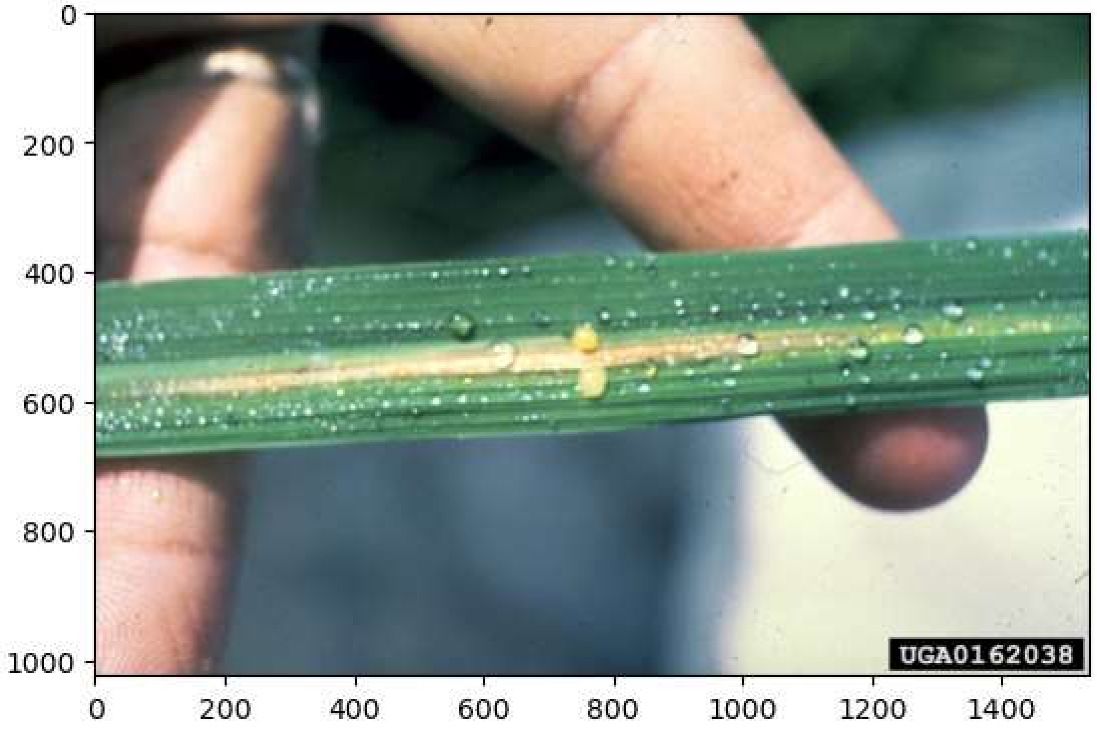
In [ ]:

In [ ]:

In [38]:
```
import numpy as np
import cv2
```

In [57]:
```
img_path = cv2.imread(os.path.join('Testing Images','bb.jpg'))
img_path = cv2.cvtColor(img_path,cv2.COLOR_BGR2RGB)
plt.imshow(img_path)
```

Out[57]:
```
<matplotlib.image.AxesImage at 0x2f08251cb20>
```



In [58]:
```
img = cv2.resize(img_path,(224,224))
img = np.reshape(img,[1,224,224,3])
```

In [59]:
```
pred = new_model.predict(img)
```

```
1/1 [==============================] - 0s 384ms/step
```

In [60]:
```
pred
```

Out[60]:
```
array([[1.000000e+00, 1.783569e-24, 0.000000e+00, 0.000000e+00,
        0.000000e+00]], dtype=float32)
```

In [61]:
```
prediction_class = class_names[np.argmax(pred)]
prediction_class
```

Out[61]:
```
'Bacterialblight'
```

In [ ]:

In [ ]: