Data Loading

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

In [*]:
```
IMAGE_SIZE = 230
BATCH_SIZE = 32
CHANNELS = 3
EPOCHS =30
```

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

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

In [*]:
```
len(dataset) #248*32=7936
```

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

In [*]:
```
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")
```

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

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

In [*]:
```
# Catching and prefeching
train_ds = train_ds.cache().shuffle(1500).prefetch(buffer_size=tf.data.AUTOTUN
val_ds = val_ds.cache().shuffle(1500).prefetch(buffer_size=tf.data.AUTOTUNE)
test_ds = test_ds.cache().prefetch(buffer_size=tf.data.AUTOTUNE)
```

Preprocessing

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

ResNet152

In [14]:
```
from tensorflow.keras.applications import ResNet152
```

In [15]:
```
resnet152 = ResNet152(input_shape=(IMAGE_SIZE,IMAGE_SIZE,CHANNELS),weights='im
```

In [16]:
```
# Don't train existing weights
for layer in resnet152.layers:
    layer.trainable = False
```

In [17]:
```
x = tf.keras.layers.Flatten()(resnet152.output)
```

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

In [19]:
```
model = tf.keras.Model(inputs=resnet152.input, outputs=prediction)
```

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

```
 conv1_bn (BatchNormalization)  (None, 115, 115, 64  256         ['conv1_c
onv[0][0]']
                                )

 conv1_relu (Activation)        (None, 115, 115, 64  0           ['conv1_b
n[0][0]']
                                )

 pool1_pad (ZeroPadding2D)      (None, 117, 117, 64  0           ['conv1_r
elu[0][0]']
                                )

 pool1_pool (MaxPooling2D)      (None, 58, 58, 64)   0           ['pool1_p
ad[0][0]']

 conv2_block1_1_conv (Conv2D)   (None, 58, 58, 64)   4160        ['pool1_p
ool[0][0]']

 conv2_block1_1_bn (BatchNormal  (None, 58, 58, 64)  256         ['conv2_b
```

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,
    mode="auto",
    baseline=None,
    restore_best_weights=False,
)
```

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

```
Epoch 1/30
198/198 [==============================] - 1675s 8s/step - loss: 2.3888 -
accuracy: 0.9058 - val_loss: 1.0320 - val_accuracy: 0.9596
Epoch 2/30
198/198 [==============================] - 1343s 7s/step - loss: 0.6140 -
accuracy: 0.9747 - val_loss: 0.4448 - val_accuracy: 0.9727
Epoch 3/30
198/198 [==============================] - 1340s 7s/step - loss: 0.3902 -
accuracy: 0.9836 - val_loss: 0.4898 - val_accuracy: 0.9896
Epoch 4/30
198/198 [==============================] - 1470s 7s/step - loss: 0.1770 -
accuracy: 0.9911 - val_loss: 0.0893 - val_accuracy: 0.9961
Epoch 5/30
198/198 [==============================] - 1683s 9s/step - loss: 0.0621 -
accuracy: 0.9956 - val_loss: 0.0365 - val_accuracy: 0.9974
Epoch 6/30
198/198 [==============================] - 1569s 8s/step - loss: 0.2103 -
accuracy: 0.9930 - val_loss: 0.1800 - val_accuracy: 0.9909
Epoch 7/30
```

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

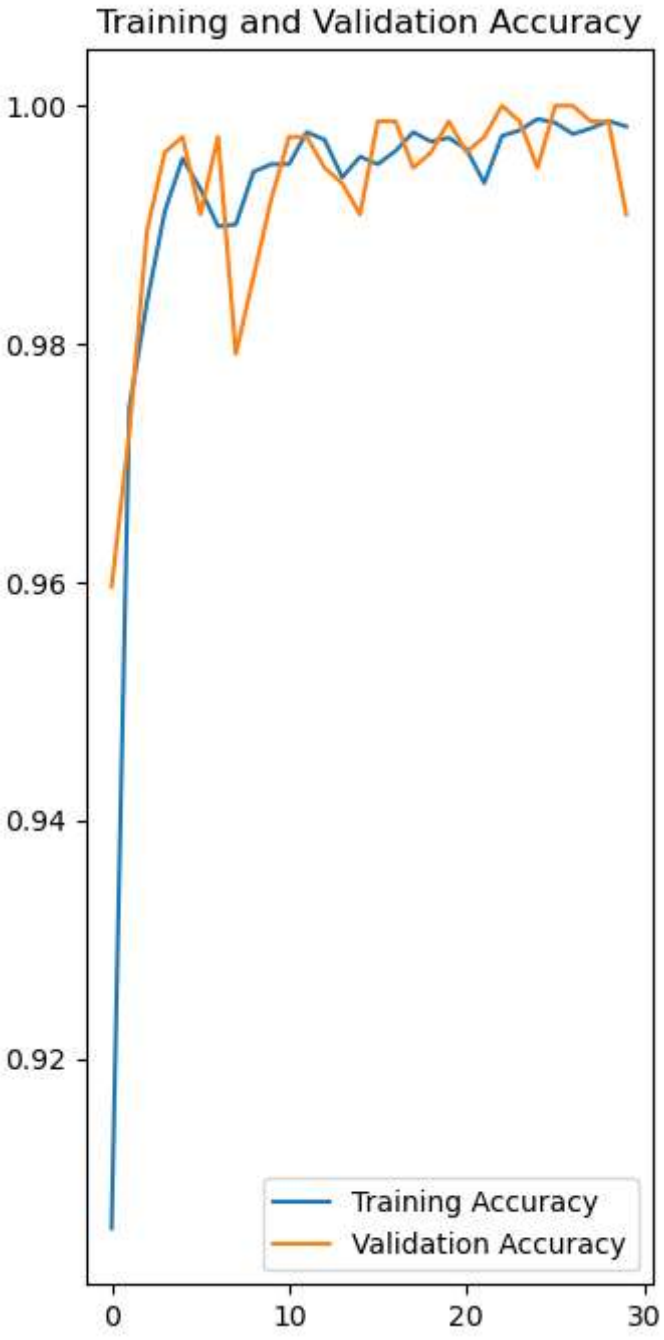
```
26/26 [==============================] - 278s 7s/step - loss: 0.8147 - accura
cy: 0.9904
```

Out[26]:
```
[0.8146988749504089, 0.9903846383094788]
```

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')
```

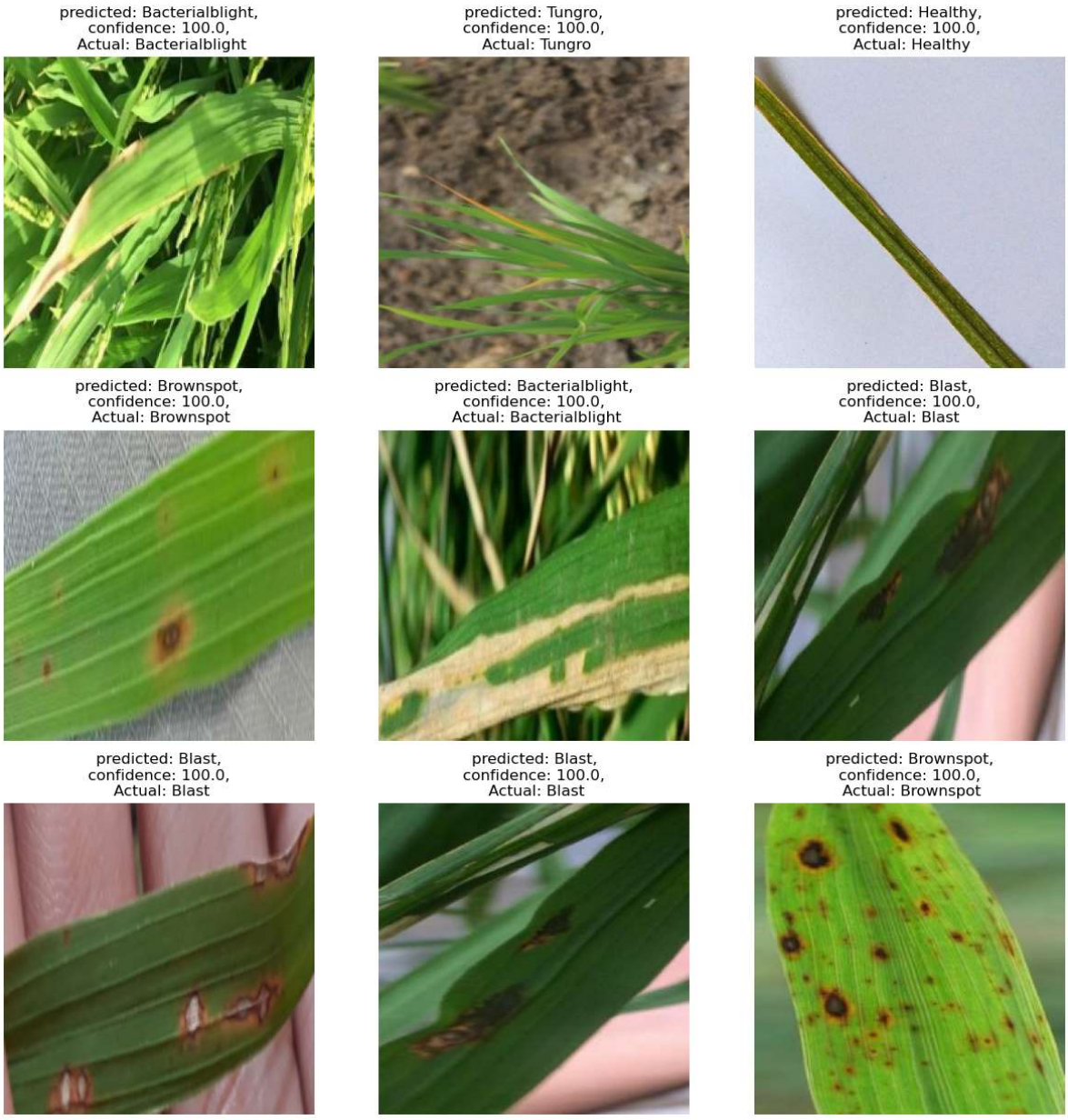
Out[28]:
```
Text(0.5, 1.0, 'Training and Validation Accuracy')
```



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 [30]:
```
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 [==============================] - 6s 6s/step
1/1 [==============================] - 0s 326ms/step
1/1 [==============================] - 0s 366ms/step
1/1 [==============================] - 0s 361ms/step
1/1 [==============================] - 0s 356ms/step
1/1 [==============================] - 0s 340ms/step
1/1 [==============================] - 0s 433ms/step
1/1 [==============================] - 0s 356ms/step
1/1 [==============================] - 0s 349ms/step
1/1 [==============================] - 0s 363ms/step
1/1 [==============================] - 0s 386ms/step
1/1 [==============================] - 0s 375ms/step
1/1 [==============================] - 0s 348ms/step
1/1 [==============================] - 0s 373ms/step
1/1 [==============================] - 0s 370ms/step
1/1 [==============================] - 0s 415ms/step
1/1 [==============================] - 0s 362ms/step
1/1 [==============================] - 0s 346ms/step
```



Saving the Trained Model

In [31]:
```
# 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 156). The
se functions will not be directly callable after loading.

INFO:tensorflow:Assets written to: new_models\8\assets

INFO:tensorflow:Assets written to: new_models\8\assets
```

In [32]:
```
# Loading saved model
```

In [ ]:
```
new_model = tf.keras.models.load_model('new_models/8')

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

Confusion Matrix and Classification Report

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

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

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

In [ ]:
```

```

New Image Classification

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

In [ ]:
```
img_path = cv2.imread(os.path.join('Testing Images','bb1.jpg'))
img_path = cv2.cvtColor(img_path,cv2.COLOR_BGR2RGB)
plt.imshow(img_path)
```

In [ ]:
```
img = cv2.resize(img_path,(230,230))
img = np.reshape(img,[1,230,230,3])
```

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

In [ ]:
```
pred
```

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

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: