In [1]:

```
# import important libraries :

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from keras import models, layers, callbacks

import warnings
warnings.filterwarnings('ignore')
tf.get_logger().setLevel('ERROR')
```

In [2]:

```
# Creating a 'earlystopping' function to trigger termination when desired
loss or accuracy is achieved.

class myCallback(callbacks.Callback):
  def on_epoch_end(self, epoch, logs={}):
    if(logs.get('accuracy') > 0.96):
      print("\nAccuracy is greater than 96% so terminating training!")
      self.model.stop_training = True

    # elif (logs.get('loss') < 0.3):
    #   print("\nLoss is less than 0.3 so terminating training!")
    #   self.model.stop_training = True

callbacks = myCallback()
```

1. LOADING DATA :

In [3]:

```
dataset = tf.keras.preprocessing.image_dataset_from_directory(
"files/train",
shuffle=True,
image_size=(256,256),
batch_size=32
)
```

```
Found 2100 files belonging to 6 classes.
```

In [4]:

```
testing = tf.keras.preprocessing.image_dataset_from_directory(
"files/validation",
shuffle=True,
image_size=(256,256),
batch_size=32
)
```

```
Found 528 files belonging to 6 classes.
```

2. DATA EXPLORATION :

In [35]:

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

Out[35]:

```
['bacterial_leaf_blight',
 'brown_spot',
 'healthy',
 'leaf_blast',
 'leaf_scald',
 'narrow_brown_spot']
```

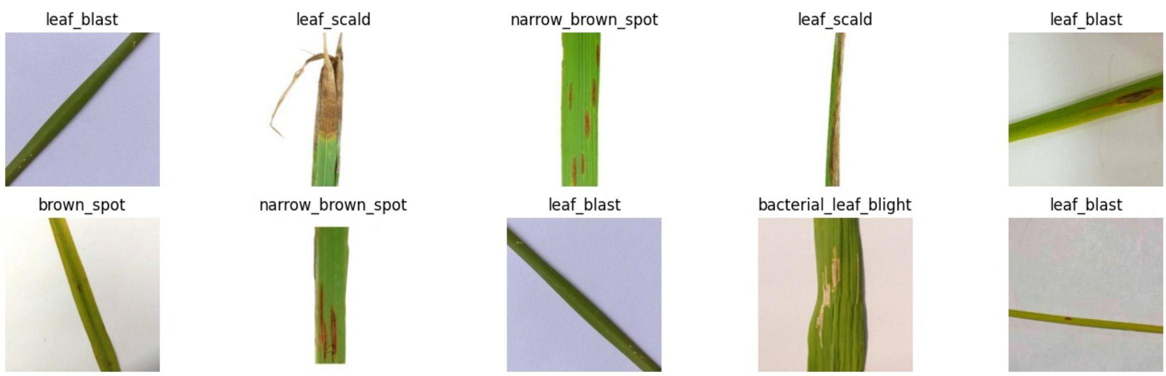
In [36]:

```
# .take() method is used to pickup a batch from the complete data, where
each batch contains 32 images (as we specified earlier).
for img,lab in dataset.take(1):
    print(img.shape)
    print(lab.numpy())
```

```
(32, 256, 256, 3)
[1 2 3 3 5 2 3 1 2 1 2 2 0 4 3 1 1 4 1 5 3 5 4 5 2 2 2 2 1 5 3 4]
```

In [40]:

```
plt.figure(figsize=(17,10))
for img,lab in dataset.take(1):
    for i in range(10):
        ax = plt.subplot(4,5,i+1),
        plt.imshow(img[i].numpy().astype("uint8"))
        plt.title(disease[lab[i]])
        plt.axis('off')
```



3. DATA PREPARATION :

In [8]:

```
len(testing)//2        # No. of batches in test data.
```

Out[8]:

8

17 batches x 1 batch with 32 image = 544 images.

In [9]:

```
# traing-test split,
valid = testing.take(8)
```

As, valid already has 8 batches from top, we can copy remaining into 'test' variable using .skip() method to keep everthing except these 8 batches from top

In [10]:

```
test = testing.skip(8)
```

In [11]:

```
len(valid), len(test), len(dataset)
```

Out[11]:

```
(8, 9, 66)
```

8 + 9 = 17 batches & 66 batches of training data, we are good to go now,

In [12]:

```
# CACHING & PREFETCHING TO MAKE THE PIPELINE HIGH PERFORMANT :

dataset.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
valid.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
test.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
```

Out[12]:

```
<PrefetchDataset element_spec=(TensorSpec(shape=(None, 256, 256, 3), dtyp
e=tf.float32, name=None), TensorSpec(shape=(None,), dtype=tf.int32, name=
None))>
```

4. PIPELINES :

- **### RESIZING & RESCALING :**

In [13]:

```
# Rescaling & Resizing data for testing and smoother training.
scale = tf.keras.Sequential([
    tf.keras.layers.experimental.preprocessing.Resizing(256,256),
    tf.keras.layers.experimental.preprocessing.Rescaling(1.0/255)
])
```

- **### DATA AUGMENTATION :**

In [14]:

```
# Adding custom images using data augmentation technique for better accur
acy.

data_augmentation = tf.keras.Sequential([
    tf.keras.layers.experimental.preprocessing.RandomFlip("horizontal_an
d_vertical"),
    tf.keras.layers.experimental.preprocessing.RandomRotation(0.2),
])
```

..now that we have scaling and data augmentation layer ready, we can proceed further with our model building part,

- **### MODEL BUILDING :**

In [15]:

```
model =  models.Sequential([
    scale,
    data_augmentation,

    # Add CNNs and maxpooling layers (trail & error work)
    layers.Conv2D(filters=32, kernel_size=(3,3), activation='relu', inpu
t_shape=(32,256,256,3)),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(filters=64, kernel_size=(3,3), activation='relu'),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(filters=64, kernel_size=(3,3), activation='relu'),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(filters=64, kernel_size=(3,3), activation='relu'),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(filters=64, kernel_size=(3,3), activation='relu'),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(filters=64, kernel_size=(3,3), activation='relu'),
    layers.MaxPooling2D((2,2)),


    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(6, activation='softmax')
])

model.build(input_shape=(32,256,256,3))
```

In [17]:

```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential (Sequential)     (32, 256, 256, 3)         0

 sequential_1 (Sequential)   (None, 256, 256, 3)       0

 conv2d (Conv2D)             (None, 254, 254, 32)      896
```

```
 max_pooling2d (MaxPooling2D  (None, 127, 127, 32)      0
 )

 conv2d_1 (Conv2D)           (None, 125, 125, 64)      18496

 max_pooling2d_1 (MaxPooling  (None, 62, 62, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 60, 60, 64)        36928

 max_pooling2d_2 (MaxPooling  (None, 30, 30, 64)       0
 2D)

 conv2d_3 (Conv2D)           (None, 28, 28, 64)        36928

 max_pooling2d_3 (MaxPooling  (None, 14, 14, 64)       0
 2D)

 conv2d_4 (Conv2D)           (None, 12, 12, 64)        36928

 max_pooling2d_4 (MaxPooling  (None, 6, 6, 64)         0
 2D)

 conv2d_5 (Conv2D)           (None, 4, 4, 64)          36928

 max_pooling2d_5 (MaxPooling  (None, 2, 2, 64)         0
 2D)

 flatten (Flatten)           (None, 256)               0

 dense (Dense)               (None, 64)                16448

 dense_1 (Dense)             (None, 6)                 390

=================================================================
Total params: 183,942
Trainable params: 183,942
Non-trainable params: 0
_________________________________________________________________
```

In [18]:

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

In [19]:

```
his = model.fit(
    dataset,
    epochs=50,
    batch_size=32,
    verbose=1,
    validation_data=valid,
    callbacks=[callbacks]
)
```

```
Epoch 1/50
66/66 [==============================] - 138s 2s/step - loss: 1.7338 - ac
curacy: 0.2186 - val_loss: 1.5420 - val_accuracy: 0.4102
Epoch 2/50
66/66 [==============================] - 129s 2s/step - loss: 1.4879 - ac
curacy: 0.3814 - val_loss: 1.4563 - val_accuracy: 0.3711
Epoch 3/50
66/66 [==============================] - 129s 2s/step - loss: 1.3997 - ac
curacy: 0.4190 - val_loss: 1.3317 - val_accuracy: 0.4570
Epoch 4/50
66/66 [==============================] - 128s 2s/step - loss: 1.3055 - ac
curacy: 0.4671 - val_loss: 1.2276 - val_accuracy: 0.4922
Epoch 5/50
66/66 [==============================] - 129s 2s/step - loss: 1.2597 - ac
curacy: 0.4757 - val_loss: 1.1558 - val_accuracy: 0.5391
Epoch 6/50
66/66 [==============================] - 130s 2s/step - loss: 1.1093 - ac
curacy: 0.5614 - val_loss: 0.8441 - val_accuracy: 0.6836
Epoch 7/50
66/66 [==============================] - 129s 2s/step - loss: 0.9472 - ac
curacy: 0.6205 - val_loss: 0.8335 - val_accuracy: 0.7031
Epoch 8/50
66/66 [==============================] - 129s 2s/step - loss: 0.8630 - ac
curacy: 0.6648 - val_loss: 0.9208 - val_accuracy: 0.6602
Epoch 9/50
66/66 [==============================] - 129s 2s/step - loss: 0.8328 - ac
curacy: 0.6814 - val_loss: 0.8582 - val_accuracy: 0.7109
Epoch 10/50
66/66 [==============================] - 129s 2s/step - loss: 0.7906 - ac
curacy: 0.6976 - val_loss: 0.7479 - val_accuracy: 0.7031
Epoch 11/50
66/66 [==============================] - 129s 2s/step - loss: 0.6673 - ac
curacy: 0.7424 - val_loss: 0.7108 - val_accuracy: 0.7305
Epoch 12/50
66/66 [==============================] - 128s 2s/step - loss: 0.6630 - ac
curacy: 0.7571 - val_loss: 0.7122 - val_accuracy: 0.7383
Epoch 13/50
66/66 [==============================] - 128s 2s/step - loss: 0.5650 - ac
curacy: 0.7967 - val_loss: 0.7484 - val_accuracy: 0.7227
Epoch 14/50
66/66 [==============================] - 128s 2s/step - loss: 0.6233 - ac
curacy: 0.7571 - val_loss: 0.5909 - val_accuracy: 0.8047
Epoch 15/50
66/66 [==============================] - 129s 2s/step - loss: 0.4751 - ac
curacy: 0.8276 - val_loss: 0.3941 - val_accuracy: 0.8594
Epoch 16/50
66/66 [==============================] - 131s 2s/step - loss: 0.5038 - ac
curacy: 0.8110 - val_loss: 0.5397 - val_accuracy: 0.8164
Epoch 17/50
66/66 [==============================] - 128s 2s/step - loss: 0.4387 - ac
curacy: 0.8362 - val_loss: 0.4231 - val_accuracy: 0.8203
Epoch 18/50
66/66 [==============================] - 128s 2s/step - loss: 0.3890 - ac
curacy: 0.8514 - val_loss: 0.4028 - val_accuracy: 0.8516
Epoch 19/50
66/66 [==============================] - 128s 2s/step - loss: 0.4052 - ac
curacy: 0.8471 - val_loss: 0.3982 - val_accuracy: 0.8750
Epoch 20/50
66/66 [==============================] - 127s 2s/step - loss: 0.4405 - ac
curacy: 0.8362 - val_loss: 0.5112 - val_accuracy: 0.8086
```

```
Epoch 21/50
66/66 [==============================] - 127s 2s/step - loss: 0.3953 - ac
curacy: 0.8571 - val_loss: 0.6883 - val_accuracy: 0.7422
Epoch 22/50
66/66 [==============================] - 127s 2s/step - loss: 0.3851 - ac
curacy: 0.8448 - val_loss: 0.5360 - val_accuracy: 0.7812
Epoch 23/50
66/66 [==============================] - 127s 2s/step - loss: 0.3567 - ac
curacy: 0.8624 - val_loss: 0.3869 - val_accuracy: 0.8672
Epoch 24/50
66/66 [==============================] - 128s 2s/step - loss: 0.3218 - ac
curacy: 0.8695 - val_loss: 0.4434 - val_accuracy: 0.8359
Epoch 25/50
66/66 [==============================] - 127s 2s/step - loss: 0.2746 - ac
curacy: 0.8929 - val_loss: 0.3243 - val_accuracy: 0.8789
Epoch 26/50
66/66 [==============================] - 127s 2s/step - loss: 0.3520 - ac
curacy: 0.8648 - val_loss: 0.6959 - val_accuracy: 0.7344
Epoch 27/50
66/66 [==============================] - 128s 2s/step - loss: 0.3719 - ac
curacy: 0.8662 - val_loss: 0.4852 - val_accuracy: 0.8164
Epoch 28/50
66/66 [==============================] - 128s 2s/step - loss: 0.3460 - ac
curacy: 0.8667 - val_loss: 0.2610 - val_accuracy: 0.8906
Epoch 29/50
66/66 [==============================] - 128s 2s/step - loss: 0.2509 - ac
curacy: 0.9076 - val_loss: 0.2995 - val_accuracy: 0.8945
Epoch 30/50
66/66 [==============================] - 128s 2s/step - loss: 0.2421 - ac
curacy: 0.9081 - val_loss: 0.3689 - val_accuracy: 0.8867
Epoch 31/50
66/66 [==============================] - 128s 2s/step - loss: 0.2174 - ac
curacy: 0.9171 - val_loss: 0.2322 - val_accuracy: 0.9102
Epoch 32/50
66/66 [==============================] - 127s 2s/step - loss: 0.2468 - ac
curacy: 0.9205 - val_loss: 0.3266 - val_accuracy: 0.8594
Epoch 33/50
66/66 [==============================] - 128s 2s/step - loss: 0.2428 - ac
curacy: 0.9105 - val_loss: 0.2795 - val_accuracy: 0.8984
Epoch 34/50
66/66 [==============================] - 128s 2s/step - loss: 0.2181 - ac
curacy: 0.9148 - val_loss: 0.5880 - val_accuracy: 0.7656
Epoch 35/50
66/66 [==============================] - 128s 2s/step - loss: 0.2323 - ac
curacy: 0.9157 - val_loss: 0.3774 - val_accuracy: 0.8555
Epoch 36/50
66/66 [==============================] - 128s 2s/step - loss: 0.2535 - ac
curacy: 0.9052 - val_loss: 0.3566 - val_accuracy: 0.8594
Epoch 37/50
66/66 [==============================] - 127s 2s/step - loss: 0.1980 - ac
curacy: 0.9257 - val_loss: 0.2805 - val_accuracy: 0.9023
Epoch 38/50
66/66 [==============================] - 128s 2s/step - loss: 0.1717 - ac
curacy: 0.9333 - val_loss: 0.2451 - val_accuracy: 0.9219
Epoch 39/50
66/66 [==============================] - 127s 2s/step - loss: 0.2399 - ac
curacy: 0.9238 - val_loss: 0.4603 - val_accuracy: 0.8320
Epoch 40/50
66/66 [==============================] - 128s 2s/step - loss: 0.3430 - ac
curacy: 0.8800 - val_loss: 0.2686 - val_accuracy: 0.9023
Epoch 41/50
```

```
Epoch 41/50
66/66 [==============================] - 127s 2s/step - loss: 0.2324 - ac
curacy: 0.9138 - val_loss: 0.2644 - val_accuracy: 0.9180
Epoch 42/50
66/66 [==============================] - 128s 2s/step - loss: 0.1429 - ac
curacy: 0.9476 - val_loss: 0.2914 - val_accuracy: 0.9023
Epoch 43/50
66/66 [==============================] - 128s 2s/step - loss: 0.1581 - ac
curacy: 0.9457 - val_loss: 0.1703 - val_accuracy: 0.9453
Epoch 44/50
66/66 [==============================] - 128s 2s/step - loss: 0.1519 - ac
curacy: 0.9443 - val_loss: 0.2806 - val_accuracy: 0.9102
Epoch 45/50
66/66 [==============================] - 128s 2s/step - loss: 0.1838 - ac
curacy: 0.9300 - val_loss: 0.1843 - val_accuracy: 0.9297
Epoch 46/50
66/66 [==============================] - 127s 2s/step - loss: 0.1220 - ac
curacy: 0.9538 - val_loss: 0.1920 - val_accuracy: 0.9414
Epoch 47/50
66/66 [==============================] - 128s 2s/step - loss: 0.1488 - ac
curacy: 0.9538 - val_loss: 0.2497 - val_accuracy: 0.9062
Epoch 48/50
66/66 [==============================] - 126s 2s/step - loss: 0.2026 - ac
curacy: 0.9205 - val_loss: 0.2142 - val_accuracy: 0.9297
Epoch 49/50
66/66 [==============================] - 124s 2s/step - loss: 0.1327 - ac
curacy: 0.9514 - val_loss: 0.2728 - val_accuracy: 0.9023
Epoch 50/50
66/66 [==============================] - ETA: 0s - loss: 0.1119 - accurac
y: 0.9614
Accuracy is greater than 96% so terminating training!
66/66 [==============================] - 124s 2s/step - loss: 0.1119 - ac
curacy: 0.9614 - val_loss: 0.3542 - val_accuracy: 0.8750
```

Evaluate on test data

In [20]:

```
score = model.evaluate(test)
```

```
9/9 [==============================] - 4s 358ms/step - loss: 0.4027 - acc
uracy: 0.8603
```

In [41]:

```
his.history.keys()
```

Out[41]:

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

In [22]:

```
loss = his.history["loss"]
val_loss = his.history["val_loss"]
acc = his.history["accuracy"]
val_acc = his.history["val_accuracy"]
```

In [42]:

```
len(acc), len(val_acc)
```

Out[42]:

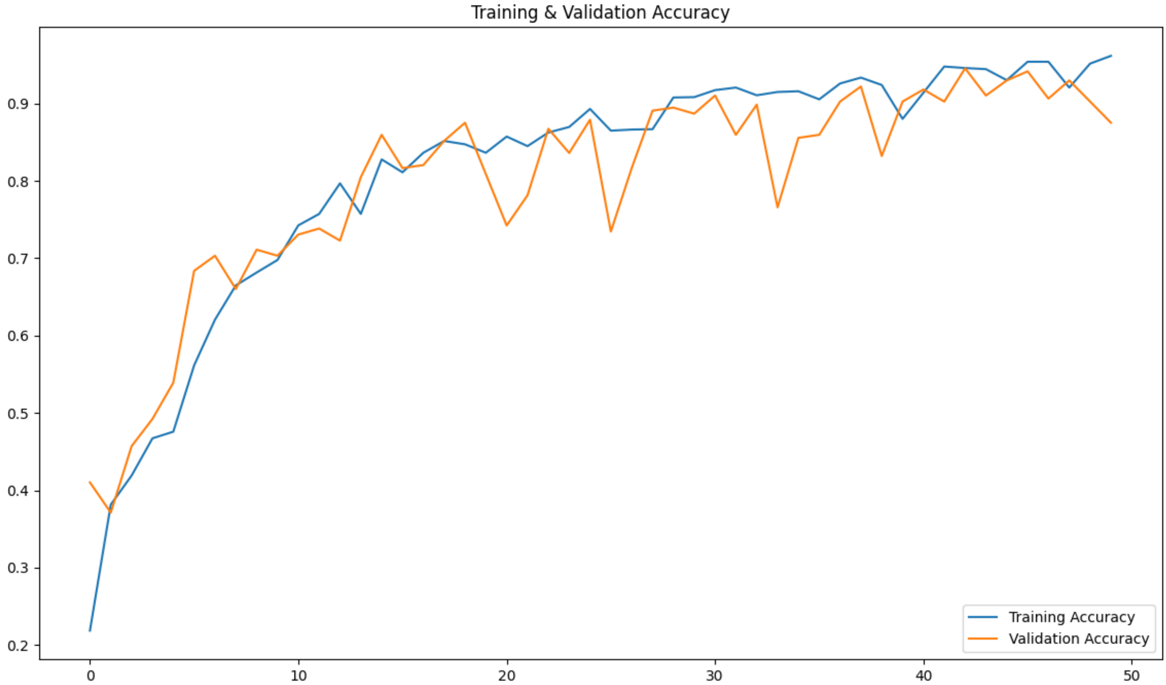
```
(50, 50)
```

In [43]:

```
plt.figure(figsize=(31,8))
plt.subplot(1,2,1)
plt.plot(range(50), acc, label='Training Accuracy')
plt.plot(range(50), val_acc, label='Validation Accuracy')
plt.legend(loc="lower right")
plt.title('Training & Validation Accuracy')
```

Out[43]:

```
Text(0.5, 1.0, 'Training & Validation Accuracy')
```



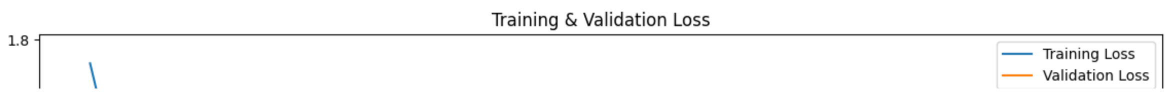
This graph shows that even increasing the epoch does not make much difference to the accuracy. Let's look at the graph of loss and conclude.

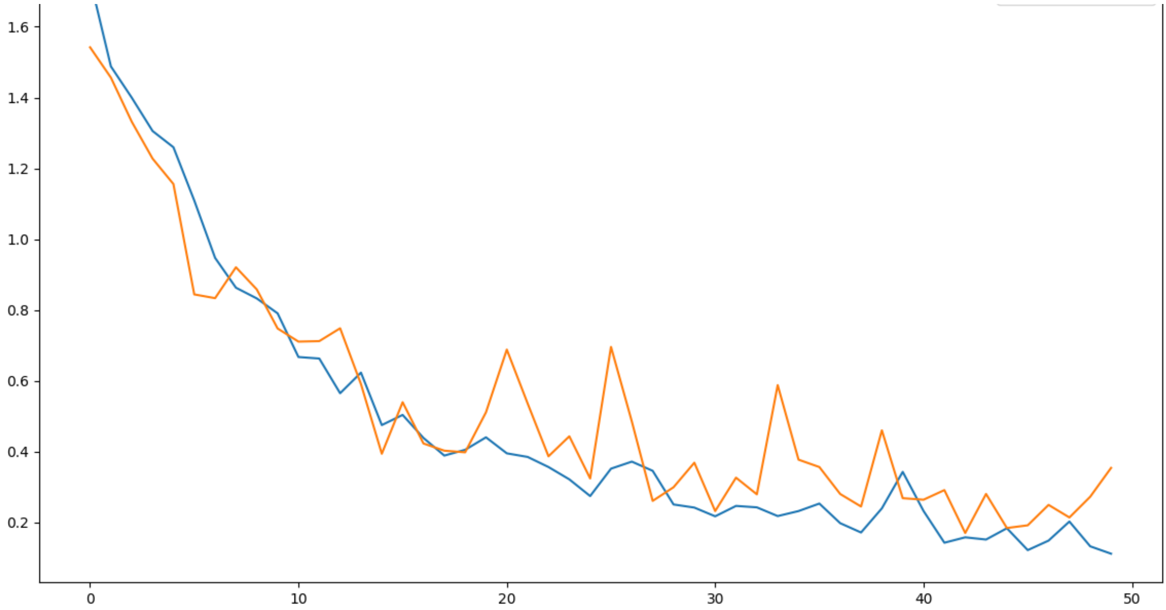
In [48]:

```
plt.figure(figsize=(31,8))
plt.subplot(1,2,1)
plt.plot(range(50), loss, label='Training Loss')
plt.plot(range(50), val_loss, label='Validation Loss')
plt.legend(loc="upper right")
plt.title('Training & Validation Loss')
```

Out[48]:

```
Text(0.5, 1.0, 'Training & Validation Loss')
```





It is possible that the loss would have reduced further, but 0.112 sounds great to me.

- **### TESTING :**

In [49]:

```
batch1 =test.take(1)
```

In [55]:

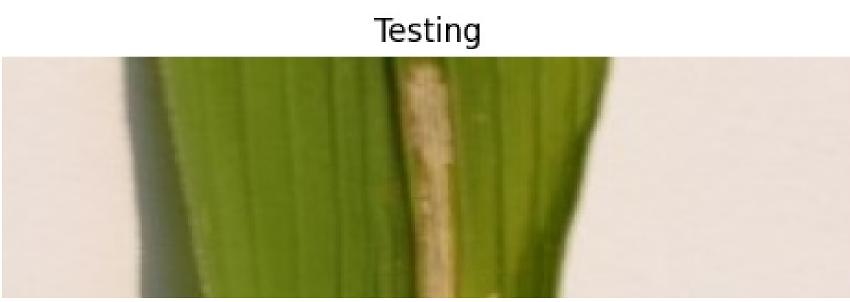
```
plt.figure(figsize=(6,6))
for img, clf in batch1:
    random_img = img[0].numpy().astype('uint8')
    typ = clf[0].numpy()

    plt.imshow(random_img)
    plt.axis('off')
    plt.title("Testing")
    prediction = model.predict(img)

    print("\nTrue :",disease[typ])
    print("Predicted :",disease[np.argmax(prediction[0])])
```

```
1/1 [==============================] - 1s 761ms/step

True : bacterial_leaf_blight
Predicted : bacterial_leaf_blight
```



In [56]:

```
def pred(img):
    array = tf.keras.preprocessing.image.img_to_array(img)
    array = tf.expand_dims(array,0)

    prediction = model.predict(array)

    clf = disease[np.argmax(prediction[0])]
    conf = round(100 * (np.max(prediction[0])), 2)
    return clf, conf
```

In [59]:

```
plt.figure(figsize=(12,15))
for img, lab in batch1:
    for i in range(9):
        ax = plt.subplot(3,3, i+1)
        plt.imshow(img[i].numpy().astype('uint8'))

        prediction, confidence = pred(img[i].numpy())
        true = disease[lab[i]]
        plt.title(f"\nTrue: {true}\nPredicted: {prediction}\nConfidence:
{confidence}%\n")
        plt.axis('off')
```

```
1/1 [==============================] - 0s 75ms/step
1/1 [==============================] - 0s 58ms/step
1/1 [==============================] - 0s 52ms/step
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 54ms/step
1/1 [==============================] - 0s 52ms/step
1/1 [==============================] - 0s 42ms/step
1/1 [==============================] - 0s 47ms/step
1/1 [==============================] - 0s 54ms/step
```

True: narrow_brown_spot
Predicted: narrow_brown_spot

True: leaf_scald
Predicted: leaf_scald

True: healthy
Predicted: healthy

In [60]:

```
import os
model_version= max([float(i) for i in os.listdir("models") ]) + 0.1
print(f"\nCurrent Version : {model_version-0.1}\nRun below cell to create version : {model_version}")
```

```
Current Version : 0.1
Run below cell to create version : 0.2
```

SAVE MODEL :

In [61]:

```
# model_version = 0.1
model.save(f"models/{model_version}")
print(f"Version {model_version} Created Successfully.")
```

```
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 7). These functions will not be directly callable after loading.
```

```
Version 0.2 Created Successfully.
```

In [ ]: