Transfer Learning methods on Rice Leaf Disease small dataset

Ryan Chui

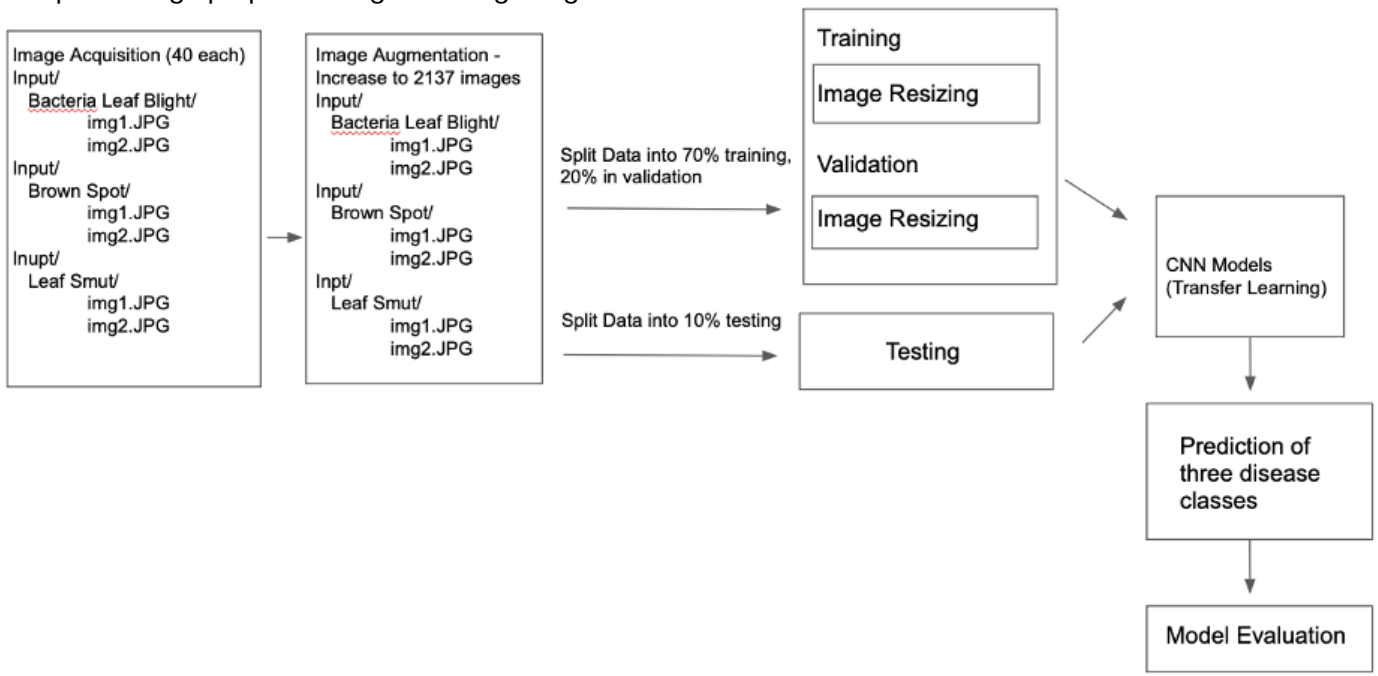
Introduction

Rice leaf suffers from several bacterial, viral, or fungal diseases and these diseases reduce rice production significantly. To sustain rice demand for a vast population globally, rice leaves related diseases often pose threats to the sustainable production of rice that affect many farmers around the world. Having farmers routinely check on rice plants daily would be costly, prone to human error, cause damage to rice plants, and many other factors could end up causing more harm than good. As a result, this is an impossible task for human look at each plant individually and examine the plants due to the vast sizes of the farms. Therefore, early diagnosis and appropriate treatments for rice leaf infection are crucial in facilitating healthy growth of rice plants due to rapid market supply and demand from customers. Thus, the aim of the project is to develop convolution neural network techniques on image classification to classify three disease classes for the rice leaf dataset that be transformed into a real-time application for farmers, without recognizing and performing inspection manually.

Data summary and pre-processing steps

The data used for this project is Rice Diseases Image Dataset from UCI reprocitory. This consists of 120 images of rice leaves with three disease classes: Bacterial leaf blight (40), Brown Spot (40), or Leaf smut (40). The dataset is gathered from a rice field in a village called Shertha in Gujarat, India, and all labels are disease infected. The leaves were captured against a white background before being photographed, and thus the dataset have a mixture of raw and processed images. While the dataset is too small and does not have enough samples, I plan to perform data augmentation to increase the dataset to 2137. Data augmentation plays a crucial role in improvising classification accuracy with the ability to acquire new training samples. I believe applying data augmentation techniques including flipping, rotating, cropping, color jittering, edge enhancement, zooming, gray scaling or re-scaling could increase model performance. All the images will need to be resized to fixed dimension of 224 pixels by 224 pixels. When the number of training samples are small, the model learns from noises or unwanted details from training samples. This negatively impacts the performance model on new samples. To overcome the challenges, data augmentation generates additional training data from the existing samples by augmenting them using random transformations which would yield quality images. This method not only reduces operational costs, but also creates variations that the model may see in real world. As a result, augmentation can be useful to reduce overfitting from adding more training data, and thereby generalize model better. Once I gathered 2137 images, I will convert into an array and visualize first few images of each class from the training folder which store images into pixel values, and the images will be labeled into its corresponding classes as 0, 1, or 2. After preprocessing steps, I divide 70% of the data for training and 20% for validation and 10% for testing before performing any classification technique and performance evaluation. The workflow of the proposed work is display in below.

Below are the steps in image preprocessing and image augmentation.



In [1]:
```
import pandas as pd
import numpy as np
import os

from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
from sklearn.metrics import confusion_matrix
import itertools
from sklearn.metrics import classification_report
import shutil

import matplotlib.pyplot as plt
import numpy as np
import cv2
import PIL
import tensorflow as tf

from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import models
from tensorflow.keras import layers
from tensorflow.keras import optimizers
import matplotlib.pyplot as plt
%matplotlib inline
from keras.preprocessing import image

IMAGE_HEIGHT = 224
IMAGE_WIDTH = 224
IMAGE_CHANNELS = 3
```

In [2]:
```
path_root = '/Users/ryan.chui/Downloads/rice_leaf_diseases_augmentated'
```

In [3]:
```
import splitfolders

# Split with a ratio.
# To only split into training and validation set, set a tuple to `ratio`, i.e, `(.8, .2)`.
#Train, val, test
splitfolders.ratio(path_root, output="'/Users/ryan.chui/Downloads/rice_leaf_diseases_augmentated_folder'
                   seed=42, ratio=(.7, .2, .1),
                   group_prefix=None) # default values
```

```
Copying files: 0 files [00:00, ? files/s]
```

In [4]:
```
# # Split val/test with a fixed number of items e.g. 100 for each set.
# # To only split into training and validation set, use a single number to `fixed`, i.e., `10`.
# # enable oversampling of imbalanced datasets, works only with fixed
# splitfolders.fixed(path_root, output="'/Users/ryan.chui/Downloads/rice_leaf_diseases_augmentated_folde
#                    seed=42, fixed=(224, 224),
#                    oversample=False, group_prefix=None)
```

In [5]:
```
data_dir = '/Users/ryan.chui/Downloads/rice_leaf_diseases_augmentated_folder'
train_dir = data_dir + '/train'
valid_dir = data_dir + '/val'
test_dir = data_dir + '/test'
```

In [6]:
```
leaf_smut_list = \
os.listdir('/Users/ryan.chui/Downloads/rice_leaf_diseases_augmentated_folder/train/Leaf Smut')
brown_spot_list = \
os.listdir('/Users/ryan.chui/Downloads/rice_leaf_diseases_augmentated_folder/train/Brown spot')
bacterial_leaf_blight_list = \
os.listdir('/Users/ryan.chui/Downloads/rice_leaf_diseases_augmentated_folder/train/Bacterial leaf blight

print(len(leaf_smut_list))
print(len(brown_spot_list))
print(len(bacterial_leaf_blight_list))
```

```
541
537
539
```

In [7]:
```
df_leaf_smut = \
os.listdir('/Users/ryan.chui/Downloads/rice_leaf_diseases_augmentated_folder/val/Leaf Smut')
df_brown_spot = \
os.listdir('/Users/ryan.chui/Downloads/rice_leaf_diseases_augmentated_folder/val/Brown spot')
df_bacterial_leaf_blight = \
os.listdir('/Users/ryan.chui/Downloads/rice_leaf_diseases_augmentated_folder/val/Bacterial leaf blight')

print(len(df_leaf_smut))
print(len(df_brown_spot))
print(len(df_bacterial_leaf_blight))
```

```
369
370
370
```

In [8]:
```
df_leaf_smut_test = \
os.listdir('/Users/ryan.chui/Downloads/rice_leaf_diseases_augmentated_folder/test/Leaf Smut')
df_brown_spot_test = \
os.listdir('/Users/ryan.chui/Downloads/rice_leaf_diseases_augmentated_folder/test/Brown spot')
df_bacterial_leaf_blight_test = \
os.listdir('/Users/ryan.chui/Downloads/rice_leaf_diseases_augmentated_folder/test/Bacterial leaf blight'

print(len(df_leaf_smut_test))
print(len(df_brown_spot_test))
print(len(df_bacterial_leaf_blight_test))
```

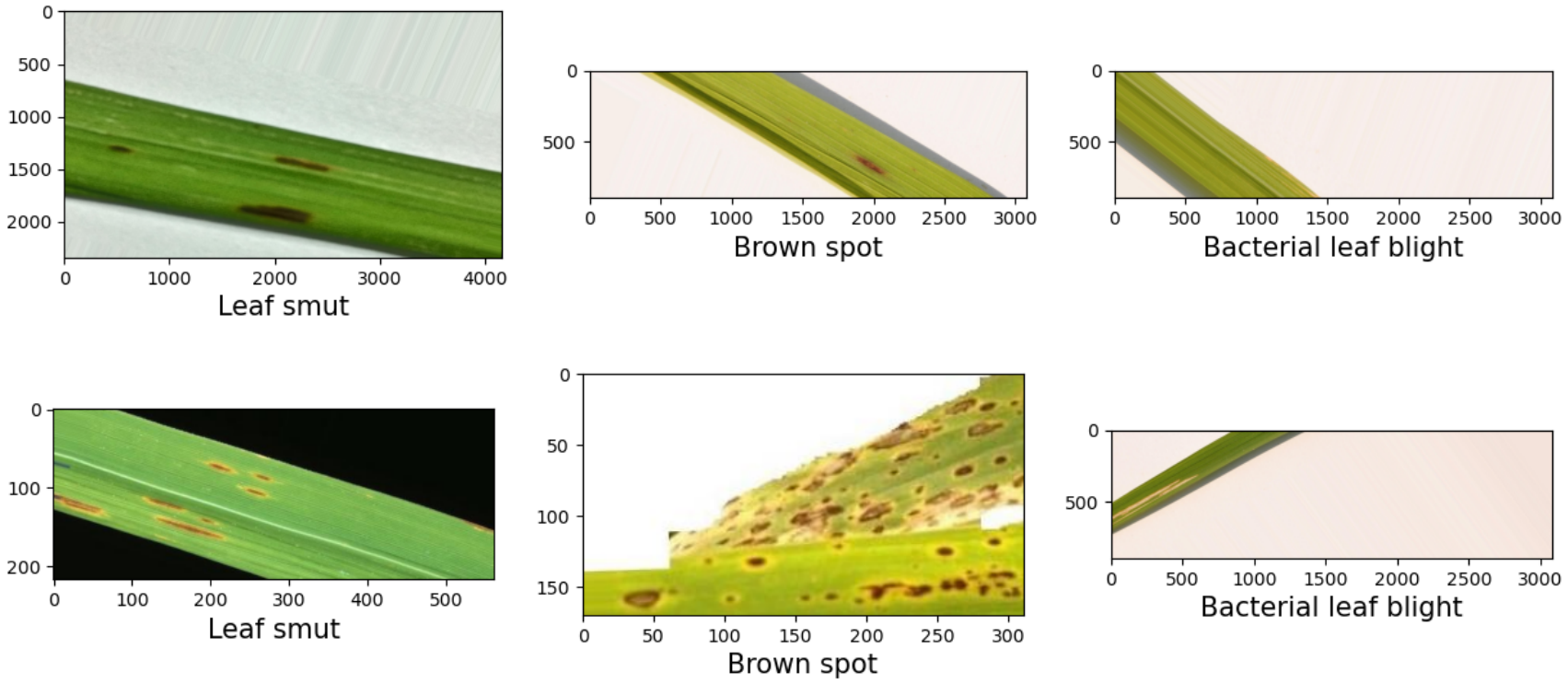
```
224
224
224
```

In [9]:
```
num_train_samples = len(leaf_smut_list) + len(brown_spot_list) + len(bacterial_leaf_blight_list)
num_val_samples = len(df_leaf_smut) + len(df_brown_spot) + len(df_bacterial_leaf_blight)
num_test_samples = len(df_leaf_smut_test) + len(df_brown_spot_test) + len(df_bacterial_leaf_blight_test)

print("Total number of training samples: ", num_train_samples)
print("Total number of validation samples: ", num_val_samples)
print("Total number of testing samples: ", num_test_samples)
```

```
Total number of training samples:  1617
Total number of validation samples:  1109
Total number of testing samples:  672
```

In [10]:
```
# set up the canvas for the subplots
plt.figure(figsize=(15,15))

# Image 1
plt.subplot(1,3,1)   # 1 row and 3 columns
image = plt.imread('/Users/ryan.chui/Downloads/rice_leaf_diseases_augmentated_folder/train/Leaf smut/DSC
plt.imshow(image)
plt.xlabel('Leaf smut', fontsize=15)

# Image 2
plt.subplot(1,3,2)   # 1 row and 3 columns
image = plt.imread('/Users/ryan.chui/Downloads/rice_leaf_diseases_augmentated_folder/train/Brown spot/DS
plt.imshow(image)
plt.xlabel('Brown spot', fontsize=15)

# Image 3
plt.subplot(1,3,3)   # 1 row and 3 columns
image = plt.imread('/Users/ryan.chui/Downloads/rice_leaf_diseases_augmentated_folder/train/Bacterial lea
plt.imshow(image)
plt.xlabel('Bacterial leaf blight', fontsize=15)

# set up the canvas for the subplots
plt.figure(figsize=(15,15))

# Image 1
plt.subplot(1,3,1)   # 1 row and 3 columns
image = plt.imread('/Users/ryan.chui/Downloads/rice_leaf_diseases_augmentated_folder/train/Leaf smut/DSC
plt.imshow(image)
plt.xlabel('Leaf smut', fontsize=15)

# Image 2
plt.subplot(1,3,2)   # 1 row and 3 columns
image = plt.imread('/Users/ryan.chui/Downloads/rice_leaf_diseases_augmentated_folder/train/Brown spot/DS
plt.imshow(image)
plt.xlabel('Brown spot', fontsize=15)

# Image 3
plt.subplot(1,3,3)   # 1 row and 3 columns
image = plt.imread('/Users/ryan.chui/Downloads/rice_leaf_diseases_augmentated_folder/train/Bacterial lea
plt.imshow(image)
plt.xlabel('Bacterial leaf blight', fontsize=15)
```

```
plt.show()
```



Transfer Learning is a popular approach in deep learning, where the pre-trained models are used as the starting point on computer vision given the vast compute and time resources required to develop neural network models. When training specifically on smaller dataset, transfer learning becomes particulary useful. It requires a model that has been pre-trained on a robust source task which can be easily adapted to solve a smaller target task (ie: 120 images), so that the learning can be easily accessed through Keras API, and fine-tuning a portion of pre-trained layers can boost model performance significantly.

I am interested in understanding how fine-tuning may affect the accuracy from a small dataset which is differed from Imagenet. Imagenet consists of 1,219,167 training images and 1000 object classes [2] in 2021, and the pre-trained model can be used to preprocess images and extract relevant features from integrating into a new model for these 2137 images. Therefore, I plan to use a pre-trained feature extraction model consists of learning rate, number of parameter, convolutional filters instead of initializing a model with random weights and train from scratch. Additionally, Keras provides convenient accesses to top performance models on image recognition tasks for Mobilenet, VGG19, ResNet.

Create the dataset using data augmentation techniques and split it into training and validation Sets

Generating a dataset from image files to store on disk is simple and fast. First, an instance of an ImageDataGenerator is created as train_dataset. The images will flow from a specific directory for the training process via train_dir. Some hyperparameters are included such as the input image size and the class_mode. Next, I decide to use Keras's flow_from_directory() function to tell my model where to get the images from and what to do with them. I store the images in a nested file structure that splits the data into training, validatiion, and testing sets, then separates the images by 3 classes. The Keras deep learning library provides the ability to use data augmentation automatically when training a model. The ImageDataGenerator class supports a number of pixel scaling methods, as well as a range of data augmentation techniques (ie: Horizontal and vertical shifting, flipping, Random rotation, zoom, and brightness). To ensure the training and validation sets don't overlap, I set the seeds to match each other.

Model Architecture - MobileNet

All images are resized to 224x224x3, and I decide to usse 1617 training images for training, 1109 validation images for validation. I choose to fine tune a Mobilenet model that was pre-trained on imagenet. Next, I choose an Adam optimizer, categorical crossentropy loss and set to learn at a constant learning rate of 0.001. I use the pre-processing method that was applied to the imagenet images that were used to pre-train Mobilenet. Image augmentation will help to reduce overfitting, improve model performance and help the model generalize better. I set the parameter pooling to 'Max' so that MobileNet outputs a vector that can directly feed into a dense layer. In the code, I explore removing the last two layers that are unused of the neural network from MobileNet and add another two custom layers.

The overall architecture of the Mobilenet consists of 30 layers with convolutional layer with stride 2, depthwise layer, pointwise layer that will double the number of channels, a depthwise layer with stride 2 and pointwise layer that will double the number of channels [1].

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Below are the steps in MobilNet implementation in Python.

In [11]:
```
#  data augmentation using ImageDataGenerator
from keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(rescale=1./255,
    rotation_range=60, width_shift_range=0.2,
    height_shift_range=0.2, shear_range=0.2,
    zoom_range=0.2,horizontal_flip=True,
    fill_mode="nearest", validation_split=0.3)


batch_size = 32
IMAGE_HEIGHT, IMAGE_WIDTH = 224, 224

train_dataset = datagen.flow_from_directory(
    train_dir,
    target_size=(IMAGE_HEIGHT, IMAGE_WIDTH),
    batch_size=batch_size,
    class_mode='categorical',
    subset='training',
    seed = 42)

val_dataset = datagen.flow_from_directory(
    valid_dir,
    target_size=(IMAGE_HEIGHT, IMAGE_WIDTH),
    batch_size=batch_size,
    class_mode='categorical',
    seed = 42)

test_gen = ImageDataGenerator(rescale=1./255)
# test_dir = '/Users/ryan.chui/Downloads/rice_leaf_diseases_augmentated/Test'

test_set = test_gen.flow_from_directory(
    test_dir,
    class_mode = 'categorical',
    target_size = (IMAGE_HEIGHT, IMAGE_WIDTH),
    batch_size = batch_size,
    seed = 42, shuffle = False
)
```

```
Found 1133 images belonging to 3 classes.
Found 1109 images belonging to 3 classes.
Found 672 images belonging to 3 classes.
```

In [12]:
```
m tensorflow.keras.models import Model, load_model
m tensorflow.keras.layers import Dense, Dropout
m tensorflow.keras.optimizers import Adam
m tensorflow.keras.metrics import categorical_accuracy
m tensorflow.keras.callbacks import (EarlyStopping, ReduceLROnPlateau, ModelCheckpoint, CSVLogger, Learn
```

In [13]:
```
from tensorflow.keras.applications.mobilenet import MobileNet

model = MobileNet(weights='imagenet', pooling='max')

# Exclude the last 2 layers of the above model.
x = model.layers[-2].output

# Create a new dense layer for predictions 3 corresponds to the number of classes
predictions = Dense(3, activation='softmax')(x)

# inputs=model.input selects the input layer, outputs=predictions refers to the dense layer we created a
model = Model(inputs=model.input, outputs=predictions)
```

```
Metal device set to: Apple M1

2023-04-23 01:41:57.542749: I tensorflow/core/common_runtime/pluggable_device/pluggable_device_factor
y.cc:305] Could not identify NUMA node of platform GPU ID 0, defaulting to 0. Your kernel may not have
been built with NUMA support.
2023-04-23 01:41:57.542861: I tensorflow/core/common_runtime/pluggable_device/pluggable_device_factor
y.cc:271] Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 0 MB memory) ->
physical PluggableDevice (device: 0, name: METAL, pci bus id: <undefined>)
```

Compiling the Model

When compiling the model, I provide the objective function (loss) that I want to minimize, optimization method (adam) and accuracy that will follow. For this Multi-class classification problem, I choose to optimize the loss using "categorical_crossentropy" instead of "binary_crossentropy" since this is not a binary classification.

For training, the 'fit()' function is used in the model with the following parameter:

train_dataset: training data

val_dataset: validation set

shuffle: change of location of data in each epoch

verbose: be able to see the outputs during the training (0-> does not show, 1-> does)

epoch: determines how many times the dataset will be trained by traversing the model

early_stop: enables me to specify and monitor the performance measure in order to end training if being triggered. Training will stop when the chosen performance measure stops improving. I would like to seek minimum validation loss and mean squared error, whereas we would seek a maximum for validation accuracy, and as a result, I choose the mode as 'min'.

Modelcheckpoint: This highlights an important concept in model selection. The notion of the “best” model during training may conflict when evaluating using different performance measures. As a reuslt, I use the ModelCheckpoint callback to save the best model based on validation accuracy. It may be interesting to know the value of the performance measure and at what epoch the model was saved, so I set "verbose" argument to 1.

In [14]:
```
# from livelossplot.inputs.keras import PlotLossesCallback
model.compile(
    Adam(learning_rate=0.001),
    loss='categorical_crossentropy',
    metrics=['accuracy'],
)

# simple early stopping
filepath = "model_mobilenet.h5"
checkpoint = ModelCheckpoint(filepath, monitor='val_accuracy', verbose=1, save_best_only=True, mode='max
# EarlyStopping
early_stop = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True, mode='min', verbo
# Save into csv file
log_fname = 'training_log.csv'
csv_logger = CSVLogger(filename=log_fname,separator=',', append=False)


callbacks_list = [checkpoint, csv_logger, early_stop]
history = model.fit(train_dataset, epochs=20,
                            validation_data=val_dataset,
                            verbose=1,
                            callbacks=callbacks_list)
```

```
Epoch 17/20
36/36 [==============================] - ETA: 0s - loss: 0.2200 - accuracy: 0.9064
Epoch 17: val_accuracy did not improve from 0.88909
36/36 [==============================] - 24s 666ms/step - loss: 0.2200 - accuracy: 0.9064 - val_loss:
0.6691 - val_accuracy: 0.8016
Epoch 18/20
36/36 [==============================] - ETA: 0s - loss: 0.2196 - accuracy: 0.9038
Epoch 18: val_accuracy did not improve from 0.88909
36/36 [==============================] - 24s 671ms/step - loss: 0.2196 - accuracy: 0.9038 - val_loss:
0.6428 - val_accuracy: 0.8079
Epoch 19/20
36/36 [==============================] - ETA: 0s - loss: 0.2202 - accuracy: 0.9047
Epoch 19: val_accuracy did not improve from 0.88909
36/36 [==============================] - 24s 673ms/step - loss: 0.2202 - accuracy: 0.9047 - val_loss:
0.7839 - val_accuracy: 0.7836
Epoch 20/20
36/36 [==============================] - ETA: 0s - loss: 0.2009 - accuracy: 0.9029
Epoch 20: val_accuracy did not improve from 0.88909
36/36 [==============================] - 24s 675ms/step - loss: 0.2009 - accuracy: 0.9029 - val_loss:
0.3978 - val_accuracy: 0.8521
```

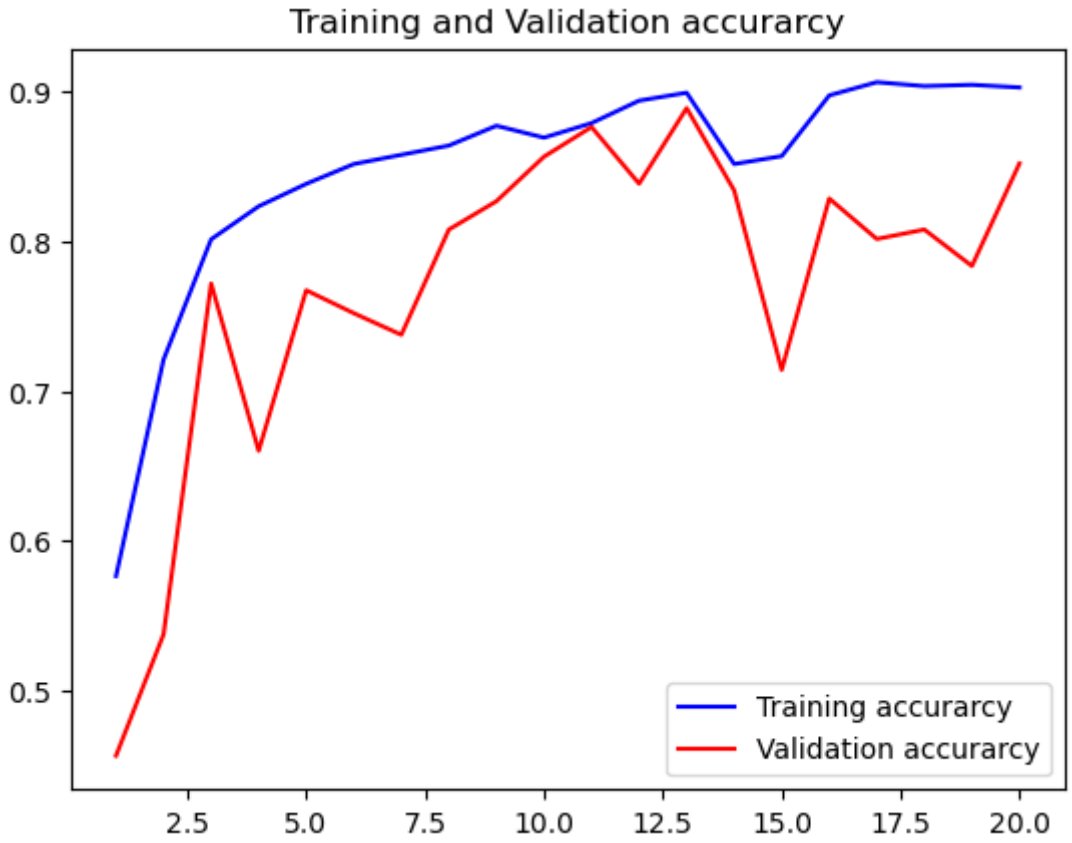
In [15]:
```
# Display the training log
train_log = pd.read_csv('training_log.csv')

train_log.head()
```

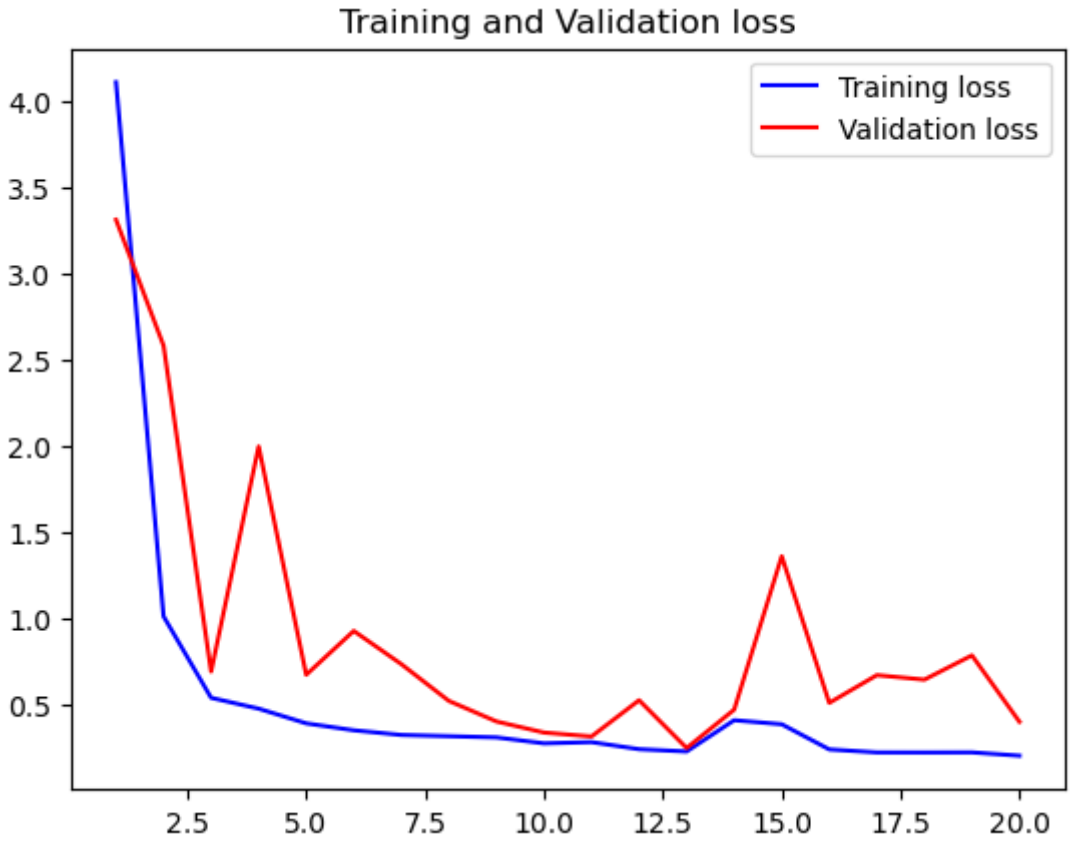
Out[15]:

	epoch	accuracy	loss	val_accuracy	val_loss
0	0	0.576346	4.114901	0.456267	3.315391
1	1	0.721094	1.010752	0.537421	2.583760
2	2	0.801412	0.537372	0.771867	0.690002
3	3	0.823478	0.474277	0.660054	1.998147
4	4	0.838482	0.388894	0.767358	0.671338

In [16]:

```
# display the loss and accuracy curves
import matplotlib.pyplot as plt

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)
#Train and validation accuracy
plt.plot(epochs, acc, 'b', label='Training accurarcy')
plt.plot(epochs, val_acc, 'r', label='Validation accurarcy')
plt.title('Training and Validation accurarcy')
plt.legend()

plt.figure()
#Train and validation loss
plt.plot(epochs, loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and Validation loss')
plt.legend()
plt.show()
```


Training and Validation accurarcy
0.9
0.8
0.7
0.6
0.5
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Training accurarcy
Validation accurarcy



The training and validation curves are being close, and I can conclude that the Model is not overfitting the data.

In [17]:
```
import seaborn as sns

def get_metrics(model, generator, steps, target_names=[]):
  """
  Function to print out confusion matrix and classification report.
  """
  target_names = ['bacterial leaf blight', 'brown spot', 'leaf smut']
  abbreviations = ['BLB', 'BS', 'LS']

  # Get predictions for data
  y_pred = model.predict_generator(generator=generator, steps=steps)
  y_pred = np.argmax(a=y_pred, axis=1)

  # Get confusion matix
  cnf_mat = confusion_matrix(y_true=generator.classes, y_pred=y_pred)
  fig, ax = plt.subplots(1)
  ax = sns.heatmap(cnf_mat, ax=ax, cmap=plt.cm.Blues, annot=True, fmt='g')
  ax.set_xticklabels(abbreviations)
  ax.set_yticklabels(abbreviations)
  plt.title('Confusion Matrix')
  plt.xlabel('Predicted Class')
  plt.ylabel('True Class')
  plt.show()

  # Get classification report
  print('Classification Report')
  print(classification_report(y_true=generator.classes, y_pred=y_pred,
                              target_names=target_names))


# Create the new validation generator for test/evaluation
val_test_gen = test_gen.flow_from_directory(
                valid_dir,
                target_size=(IMAGE_HEIGHT, IMAGE_WIDTH),
                color_mode='rgb',
                batch_size=1,
                class_mode='categorical',
                shuffle=False)
```

```
val_test_gen.classes
```

```
Found 1109 images belonging to 3 classes.
```

Out[17]:
```
array([0, 0, 0, ..., 2, 2, 2], dtype=int32)
```

Evaluate model's performance with the Validation set images (Optional)

Training set: Used for deriving the Machine Learning algorithm to capture the relationships in the data.

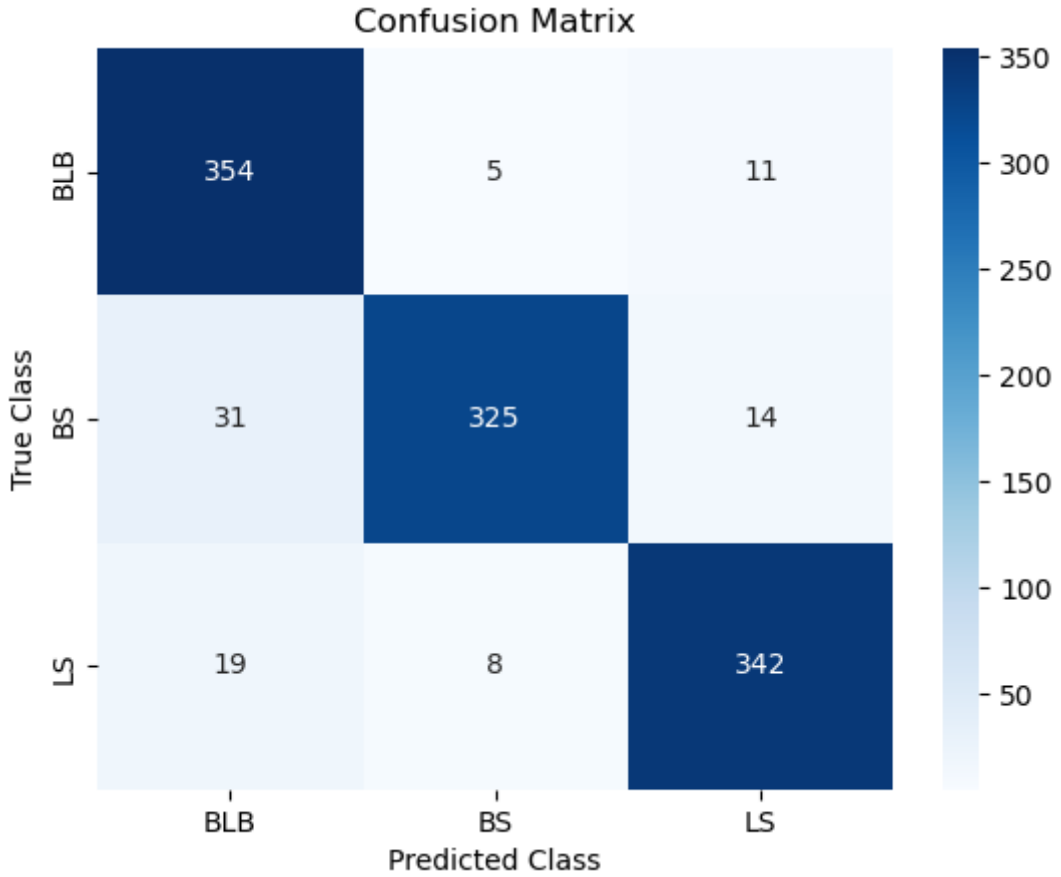
Validation set: Used for an unbiased evaluation of the model fit during hyperparameter tuning, model selection, and error analysis.

Test set or Hold-out set: Used for the final, independent evaluation with data not seen by the algorithm during the training and validation processes.

Below, I evaluate the model performance on the validation set by fitting the model and calculate its performance on the validation dataset.

In [18]:
```
# Load the saved model
model = load_model(filepath='model_mobilenet.h5')

# Get confusion matrix and classification report
get_metrics(model, generator = val_test_gen, steps= num_val_samples)
```

```
/var/folders/sc/_h6pchkd4cd_dg05d0681q2w0000gn/T/ipykernel_62432/1487657501.py:11: UserWarning: `Mode
l.predict_generator` is deprecated and will be removed in a future version. Please use `Model.predict
`, which supports generators.
  y_pred = model.predict_generator(generator=generator, steps=steps)
2023-04-23 01:50:03.714352: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:1
13] Plugin optimizer for device_type GPU is enabled.
```



```
Classification Report
                      precision    recall  f1-score   support

bacterial leaf blight       0.88      0.96      0.91       370
           brown spot       0.96      0.88      0.92       370
            leaf smut       0.93      0.93      0.93       369

             accuracy                           0.92      1109
            macro avg       0.92      0.92      0.92      1109
         weighted avg       0.92      0.92      0.92      1109
```

In [19]:
```
# Mobile Net
val_dataset.reset()

# Evaluate on Validation data
scores = model.evaluate(val_dataset)
print("%s%s: %.2f%%" % ("evaluate ",model.metrics_names[1], scores[1]*100))
scores = model.evaluate_generator(val_dataset)
print("%s%s: %.2f%%" % ("evaluate_generator ",model.metrics_names[1], scores[1]*100))
```

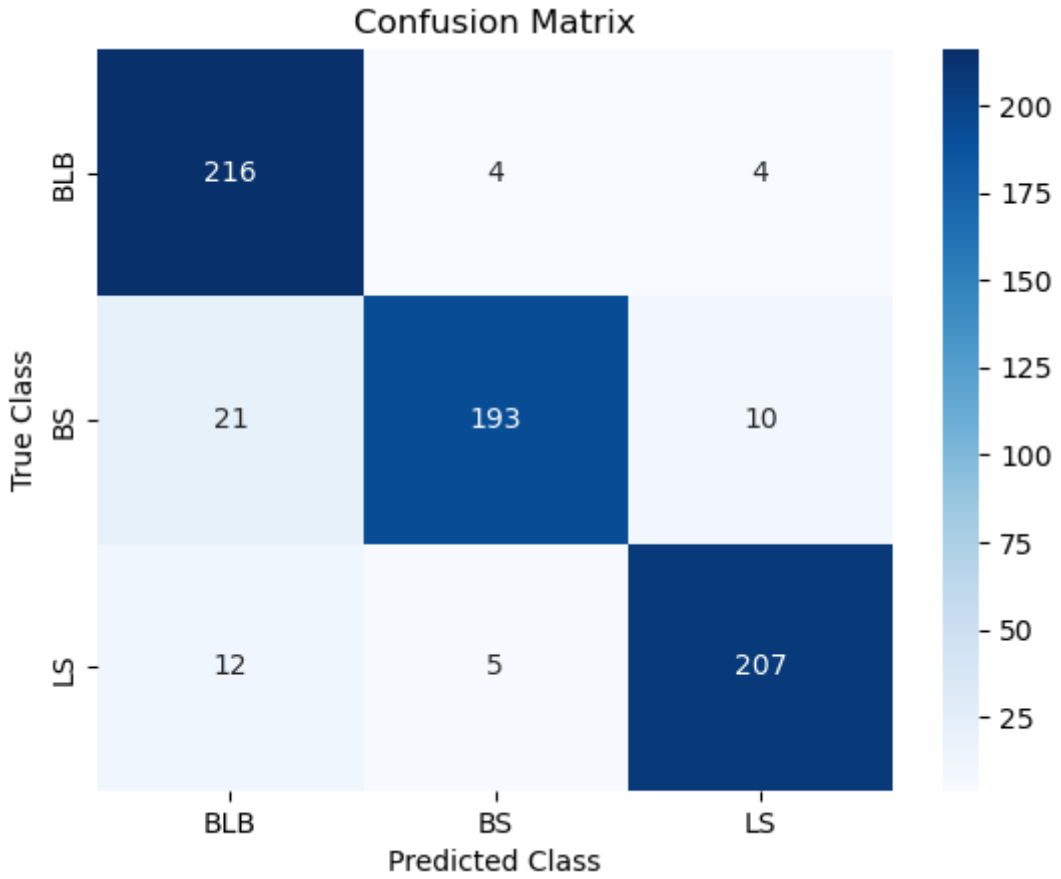
```
2023-04-23 01:50:14.067118: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:1
13] Plugin optimizer for device_type GPU is enabled.

35/35 [==============================] - 11s 283ms/step - loss: 0.2484 - accuracy: 0.8954
evaluate accuracy: 89.54%

/var/folders/sc/_h6pchkd4cd_dg05d0681q2w0000gn/T/ipykernel_62432/2230488393.py:7: UserWarning: `Model.
evaluate_generator` is deprecated and will be removed in a future version. Please use `Model.evaluate
`, which supports generators.
  scores = model.evaluate_generator(val_dataset)

evaluate_generator accuracy: 88.82%
```

Evaluate model's performance with the test set images (Test the model on independent hold-out set)

Our goal here is to use the test set only for the final evaluation. The reason for that is to test a model exclusively on unseen data that has not yet been seen by the model. I hold off the assessment with a test set until training is completed. While the model is making decisions on the evaluation set images, it can lead to information leakage. As a reuslt, the test dataset provides the gold standard used to evaluate the model and should be served once a model is completely trained.

In [20]:
```
# Create the new validation generator for test/evaluation
from sklearn.metrics import accuracy_score, f1_score, recall_score, precision_score

# Generate predictions
model.load_weights('model_mobilenet.h5') # initialize the best trained weights

true_classes = test_set.classes

model_preds_ft = model.predict(test_set)
model_pred_classes_ft = np.argmax(model_preds_ft, axis=1)

# No data augmentation for validation data
test_datagen = ImageDataGenerator(rescale=1./255)

test_gen = test_datagen.flow_from_directory(
                test_dir,
                target_size=(IMAGE_HEIGHT, IMAGE_WIDTH),
                color_mode='rgb',
                batch_size=1,
                class_mode='categorical',
                shuffle=False)

# Get confusion matrix and classification report
get_metrics(model, generator=test_gen, steps= num_test_samples)

print ('Accuracy:', accuracy_score(true_classes, model_pred_classes_ft))
print ('F1 score:', f1_score(true_classes, model_pred_classes_ft, average='macro'))
print ('Recall:', recall_score(true_classes, model_pred_classes_ft, average='macro'))
print ('Precision:', precision_score(true_classes, model_pred_classes_ft, average='macro'))
```

```
21/21 [==============================] - 3s 137ms/step
Found 672 images belonging to 3 classes.

/var/folders/sc/_h6pchkd4cd_dg05d0681q2w0000gn/T/ipykernel_62432/1487657501.py:11: UserWarning: `Mode
l.predict_generator` is deprecated and will be removed in a future version. Please use `Model.predict
`, which supports generators.
  y_pred = model.predict_generator(generator=generator, steps=steps)
```



Classification Report

	precision	recall	f1-score	support
bacterial leaf blight	0.87	0.96	0.91	224
brown spot	0.96	0.86	0.91	224
leaf smut	0.94	0.92	0.93	224
accuracy			0.92	672
macro avg	0.92	0.92	0.92	672
weighted avg	0.92	0.92	0.92	672

Accuracy: 0.9166666666666666
F1 score: 0.9165865346457654
Recall: 0.9166666666666666
Precision: 0.9198556692609783

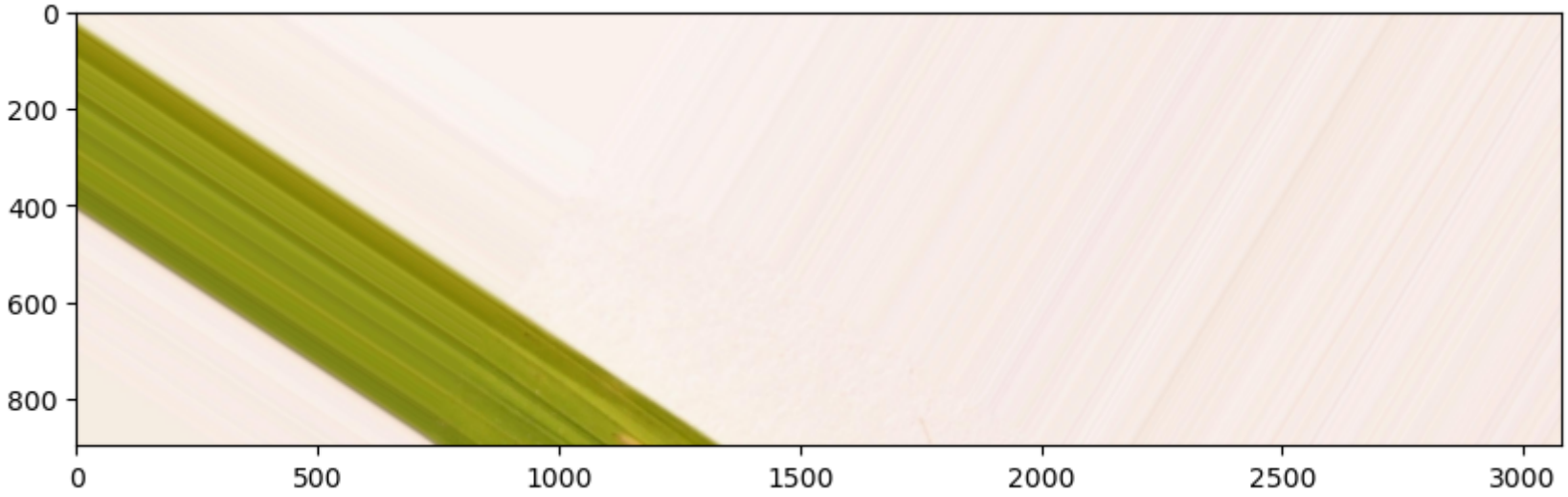
In [21]:
```
# Find the images that were misclassified
y_pred = model.predict_generator(generator = test_gen, steps= num_test_samples)
y_pred = np.argmax(y_pred, axis=1)
y_true = test_gen.classes

# Indices where misclassified
idx = np.where(np.not_equal(y_pred, y_true))
filepaths = [test_gen.filepaths[i] for i in idx[0]]

print("Number of Misclassified samples :" , len(filepaths))
filepaths[:10]
```

```
/var/folders/sc/_h6pchkd4cd_dg05d0681q2w0000gn/T/ipykernel_62432/712670774.py:2: UserWarning: `Model.p
redict_generator` is deprecated and will be removed in a future version. Please use `Model.predict`, w
hich supports generators.
  y_pred = model.predict_generator(generator = test_gen, steps= num_test_samples)

Number of Misclassified samples : 56
```

Out[21]:
```
['/Users/ryan.chui/Downloads/rice_leaf_diseases_augmentated_folder/test/Bacterial leaf blight/DSC_0_15
20.JPG',
 '/Users/ryan.chui/Downloads/rice_leaf_diseases_augmentated_folder/test/Bacterial leaf blight/DSC_0_22
05.JPG',
 '/Users/ryan.chui/Downloads/rice_leaf_diseases_augmentated_folder/test/Bacterial leaf blight/DSC_0_23
42.JPG',
 '/Users/ryan.chui/Downloads/rice_leaf_diseases_augmentated_folder/test/Bacterial leaf blight/DSC_0_25
26.JPG',
 '/Users/ryan.chui/Downloads/rice_leaf_diseases_augmentated_folder/test/Bacterial leaf blight/DSC_0_28
64.JPG',
 '/Users/ryan.chui/Downloads/rice_leaf_diseases_augmentated_folder/test/Bacterial leaf blight/DSC_0_80
22.JPG',
 '/Users/ryan.chui/Downloads/rice_leaf_diseases_augmentated_folder/test/Bacterial leaf blight/DSC_0_83
88.JPG',
 '/Users/ryan.chui/Downloads/rice_leaf_diseases_augmentated_folder/test/Bacterial leaf blight/DSC_0_93
86.JPG',
 '/Users/ryan.chui/Downloads/rice_leaf_diseases_augmentated_folder/test/Brown spot/DSC_0_1290.JPG',
 '/Users/ryan.chui/Downloads/rice_leaf_diseases_augmentated_folder/test/Brown spot/DSC_0_1349.JPG']
```

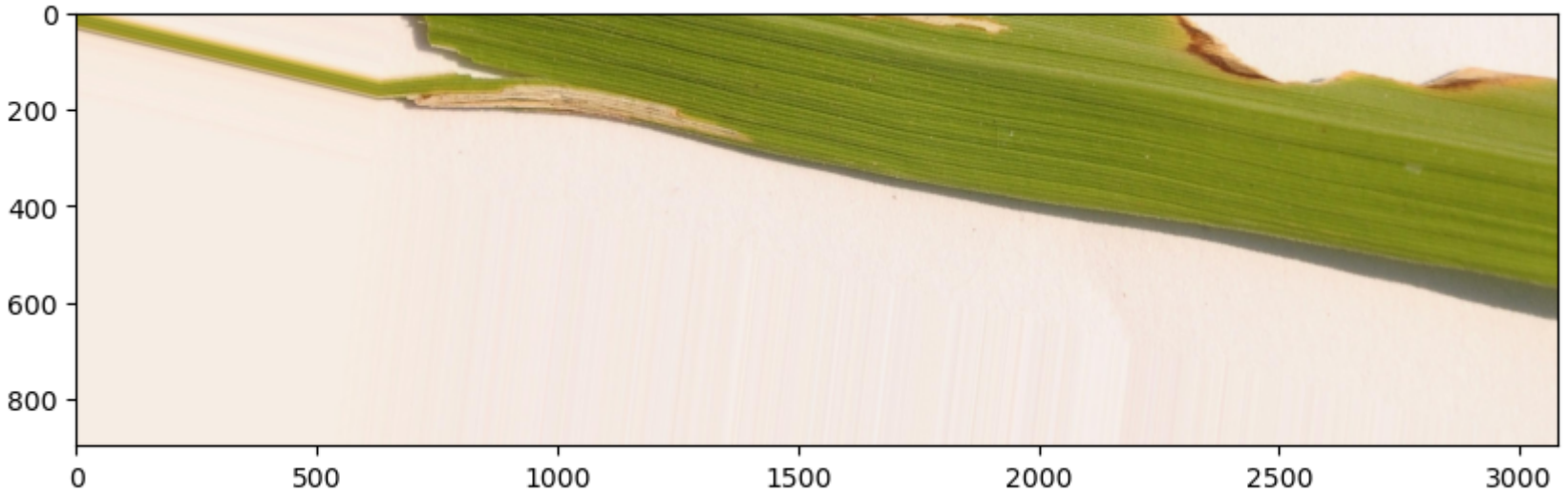
In [49]:
```
om tensorflow.keras.preprocessing import image

isplay image smaller (for checking manually)
f display_image(image_directory):
  im = image.load_img(image_directory)
  fig, ax = plt.subplots(figsize=(10,6))
  # displaying the image with figsize
  ax.imshow(im)

unction for predicting category of slected image
f testing_image(image_directory):

  # loading testing image with the target size for the image
  test_image = image.load_img(image_directory, target_size = (224,224))
  # makes sure the image is in RGB (converts all images to have only 3 color channels, png images have 4
  test_image = test_image.convert(mode='RGB')
  # converts image into an array
  test_image = tf.keras.preprocessing.image.img_to_array(test_image)
    # expands array (from converted image) with a new dimension (for classifying values)
  test_image = np.expand_dims(test_image, axis = 0)

  # making prediction based on test_image and labeling it results
  result = model.predict(x = test_image)
  # printing predictions
  print(result)

  #display image smaller (for checking manually)
  display_image(image_directory)

  # computing category weither a shape is a Bacterial leaf, Brown Spot or Leaf Smut
  if (result[0][0] > result[0][1] + result[0][2]):
      print(f'Shape is predicted as a Bacterial leaf blight with {"%.1f" % ((result[0][0])*100)}% accura
  elif (result[0][1] > result[0][0] + result[0][2]):
      print(f'Shape is predicted as a Brown Spot with {"%.1f" % ((result[0][1])*100)}% accuracy')
  elif result[0][2] > result[0][0] + result[0][1]:
      print(f'Shape is predicted as a Leaf Smut with {"%.1f" % ((result[0][2])*100)}% accuracy')

print test image classified category
sting_image('/Users/ryan.chui/Downloads/rice_leaf_diseases_augmentated_folder/test/Bacterial leaf blight/
```

```
1/1 [==============================] - 0s 19ms/step
[[0.36942053 0.62553906 0.00504041]]
Shape is predicted as a Brown Spot with 62.6% accuracy
```



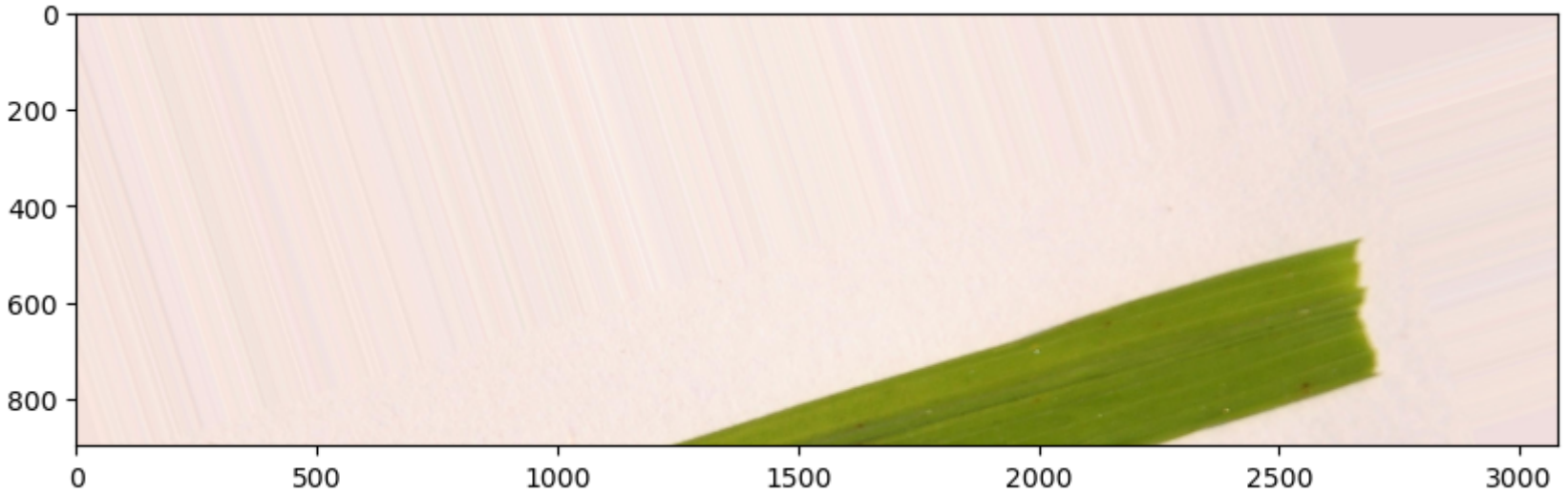
In [50]:
```
# print test image classified category
testing_image('/Users/ryan.chui/Downloads/rice_leaf_diseases_augmentated_folder/test/Bacterial leaf blig
```

```
1/1 [==============================] - 0s 21ms/step
[[0.2795414  0.7134562  0.00700238]]
Shape is predicted as a Brown Spot with 71.3% accuracy
```



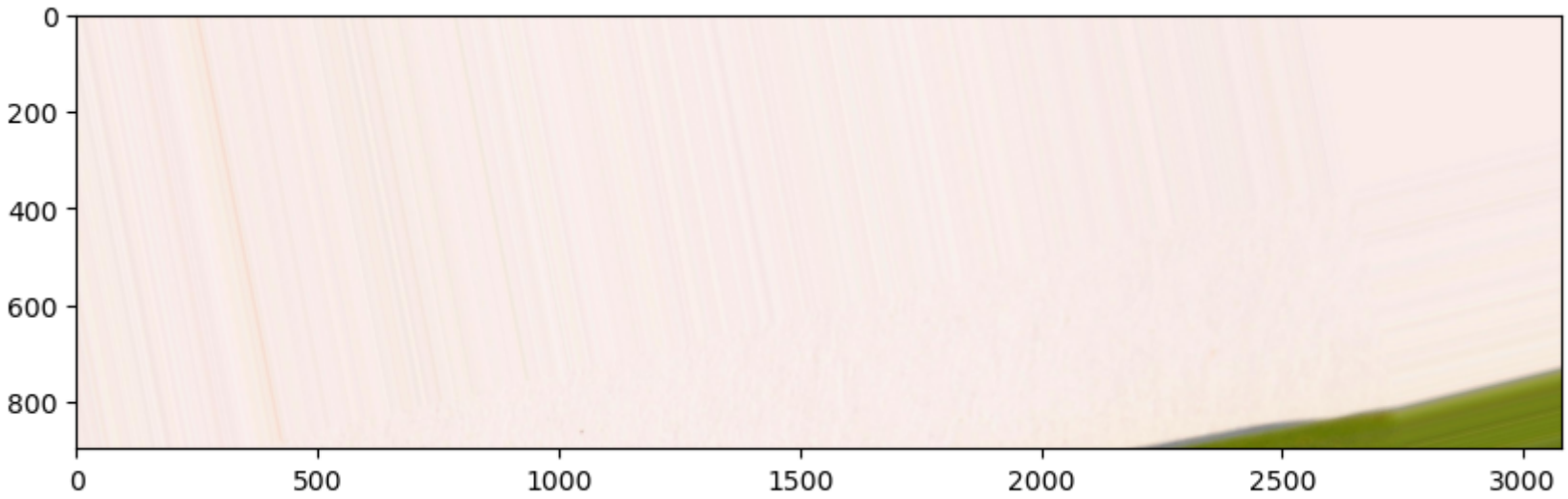
In [51]:
```
# print test image classified category
testing_image('/Users/ryan.chui/Downloads/rice_leaf_diseases_augmentated_folder/test/Brown Spot/DSC_0_12
```

```
1/1 [==============================] - 0s 26ms/step
[[0.8373418  0.1414483  0.02120997]]
Shape is predicted as a Bacterial leaf blight with 83.7% accuracy
```



In [53]:
```
testing_image('/Users/ryan.chui/Downloads/rice_leaf_diseases_augmentated_folder/test/Brown Spot/DSC_0_13
```

```
1/1 [==============================] - 0s 21ms/step
[[0.81348765 0.12513937 0.06137305]]
Shape is predicted as a Bacterial leaf blight with 81.3% accuracy
```



Above 4 images shows how well the Mobile Nnet model generalizes to unseen data, more specifically on the test dataset that was held out. Specifically, the first two images are incorrectedly predicted as Brown Spot when the true label is Bacterial Leaf Blight, and the last two images are incorrectedly predicted as Bacterial leaf blight when the true label is Brown Spot.

Conclusion

The transfer learning model using MobileNet is the best, evident in confusion matrix shows stronger diagonal and in lighter cells everywhere. Additionally, this model most commonly misclassifies Brown Spot as Leaf Smut. A total of 68 images in testing are misclassfied from 672 samples. Overall, it's a clear winner of having accuracy of 90% compared with VGG19, and Inception Net. Resnet does not give a good prediction accuracy and is excluded in this report. Mobile Net gives the best accuracy and prediction so far while I have explored VGG19 with fine-tuning, Inception Net. As a result, the effectiveness of transfer learning depends on several factors, including the type of pre-trained model used, the similarity between the original task of a pre-trained model which was trained on and the new task, the size and quality of the new dataset. Please refer Appendix about model accuracy and predictions on the pre-trained models (ie: VGG19, Inception Net).

Reference

[1] Howard, Andrew G., et al. “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications.” ArXiv.org, 17 Apr. 2017, https://arxiv.org/abs/1704.04861 (https://arxiv.org/abs/1704.04861).

[2] Solawetz, Jacob. “An Introduction to ImageNet.” Roboflow Blog, Roboflow Blog, 28 June 2021, https://blog.roboflow.com/introduction-to-imagenet/ (https://blog.roboflow.com/introduction-to-imagenet/).

[3] Simonyan, Karen, and Andrew Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition.” ArXiv.org, 10 Apr. 2015, https://arxiv.org/abs/1409.1556 (https://arxiv.org/abs/1409.1556).

[4] Nguyen, T.-H., Nguyen, T.-N., & Ngo, B.-V. (2022, October 5). A VGG-19 model with transfer learning and image segmentation for classification of Tomato Leaf Disease. MDPI. Retrieved April 1, 2023, from https://www.mdpi.com/2624-7402/4/4/56 (https://www.mdpi.com/2624-7402/4/4/56)

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: