This is a general purpose image classifier that can be applied to datasets that have the data organized as shown below. To use this code select Run all. The code will prompt the user to provide the required input data. It will take only a minute or so to enter the data and your model will be up and training. I have used this code on numerous data sets with excellent results. The model will initially run for 10 epochs. After 10 epochs you are prompted to enter H to halt training or you can enter an integer that specifies how many more epochs to run. After those epochs complete you will be again prompted to continue or halt training. The code shows the % decreasee of the validation loss at the end of each epoch. This is a good guide to use to decide to continue or halt training. As you near the bottom of the loss function the % decrease in loss becomes smaller. I typically halt training when the % decrease falls below 5%. However sometimes it is best to continue training until the loss increases. The code then automatically lowers the learning rate. This often enables the model to again start decreasing the loss. Your saved model is always returned with the weights set to those for the epoch with the lowest loss.

data directory

- train directory
  - class 0
    - image 0
    - image 1
    - ......
    - image m
  - class 1
    - image 0
    - image 1
    - ......
    - image n
  - ......
  - class m
    - image 0
    - image 1
    - ......
    - image r
- valid directory (optional if there is no validation directory validation data is created by partition of train data)
  - class 0
    - image 0
    - image 1
    - ......
    - image x
  - class 1
    - image 0

        - image 1
        - ......
        - image y
    - ......
    - class m
        - image 0
        - image 1
        - ......
        - image z
- test directory (optional if there is no test directory test data is created by partition of train data)
    - class 0
        - image 0
        - image 1
        - ......
        - image a
    - class 1
        - image 0
        - image 1
        - ......
        - image b
    - ......
    - class m
        - image 0
        - image 1
        - ......
        - image c

1. Import Needed Modules

2 Define a function to print text in color

3. Define a function to plot the number of images in dataset classes

4 Define a function to return information on the dataset

5. Read in images and create a dataframe of image paths and class labels

6. Define a function that trims a dataset for the max number of class images

7. Balance the trainning set using augmentation

8. Create train, test and validation generators

9. Create a function to show Training Image Samples

10 Create a function to calculate the F1 score metric

11. Create the Model

12. Create a custom Keras callback to continue or halt training

13. Define a function to plot the training data

14. Define a function save the training data to a csv file

15. Define a function to make predictions on the test set

16. Make predictions on test set, create Confusion Matrix and Classification Report

17. Define a function to save the trained model

18 Define the run function which runs the classifier

19 Define the code to enable to initiate the run function

20. Evaluate Model Performance

Recently Kaggle updated its docker file to load tensorflow 2.11.0. where before it loaded version 2.9.2 . However version 2.10.0 and above have a bug with respect to saving EfficientNet models. In my code if you select a medium or large model, the result is an EfficientNetB0 or B3 model is generated. So when you try to save the model you will receive a message saying the model can not be saved. If you want to save your model, uncomment the code line below which will install tensorflow version 2.9.2.

In [1]:
```
#! pip install tensorflow==2.9.2
```

Import required modules

In [2]:
```
import pandas as pd
import numpy as np
from numpy import random
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
import time
import matplotlib
import matplotlib.pyplot as plt
import cv2
import seaborn as sns
sns.set_style('darkgrid')
import shutil
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Dense, Activation,Dropout,Conv2D, MaxPooling
from tensorflow.keras.optimizers import Adam, Adamax
from tensorflow.keras.metrics import categorical_crossentropy
from tensorflow.keras import regularizers
from tensorflow.keras.models import Model
from tensorflow.keras import backend as K
import albumentations as A
import time
from tqdm import tqdm
from sklearn.metrics import f1_score
from IPython.display import YouTubeVideo
import sys
if not sys.warnoptions:
    import warnings
    warnings.simplefilter("ignore")
pd.set_option('display.max_columns', None)  # or 1000
pd.set_option('display.max_rows', None)  # or 1000
pd.set_option('display.max_colwidth', None)  # or 199
```

```
/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A Num
Py version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected v
ersion 1.24.3
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

In [3]:
```
# Seed Everything to reproduce results for future use cases
def seed_everything(seed=42):
    # Seed value for TensorFlow
    tf.random.set_seed(seed)

    # Seed value for NumPy
    np.random.seed(seed)

    # Seed value for Python's random library
    random.seed(seed)

    # Force TensorFlow to use single thread
    # Multiple threads are a potential source of non-reproducible results.
    session_conf = tf.compat.v1.ConfigProto(
        intra_op_parallelism_threads=1,
        inter_op_parallelism_threads=1
    )

    # Make sure that TensorFlow uses a deterministic operation wherever possible
    tf.compat.v1.set_random_seed(seed)

    sess = tf.compat.v1.Session(graph=tf.compat.v1.get_default_graph(), config=s
    tf.compat.v1.keras.backend.set_session(sess)

seed_everything()
```

Define a function to print text in specified rgb foreground and background colors

Add some PZAZZ to your printed output with this function

form of the call is: print_in_color(txt_msg, fore_tupple, back_tupple where:

- txt_msg is the string to be printed out
- fore_tuple is tuple of the form (r,g,b) specifying the foreground color of the text
- back_tuple is tuple of the form (r,g,b) specifying the background color of the text

In [4]:
```
# for kaggle uae (0,0,0) and (255,255,255)
def print_in_color(txt_msg,fore_tupple= (0,255,255) ,back_tupple=(0,0,0)):
    #prints the text_msg in the foreground color specified by fore_tupple with t
    #text_msg is the text, fore_tupple is foregroud color tupple (r,g,b), back_t
    rf,gf,bf=fore_tupple
    rb,gb,bb=back_tupple
    msg='{0}' + txt_msg
    mat='\33[38;2;' + str(rf) +';' + str(gf) + ';' + str(bf) + ';48;2;' + str(rb
```

```
    print(msg .format(mat))
    print('\33[0m', end='') # returns default print color to back to black
```

Define a function that plots value counts for a column in a dataframe

In [5]:
```
def plot_label_count (df, plot_title):
    column='labels'
    xaxis_label='CLASS'
    yaxis_label='IMAGE COUNT'
    vcounts=df[column].value_counts()
    labels=vcounts.keys().tolist()
    values=vcounts.tolist()
    lcount=len(labels)
    if lcount>60:
        print_in_color('The number of labels is >55, no plot will be produced')
    else:
        width=lcount * 4
        width=np.min([width, 20])
        plt.figure(figsize=(width,5))
        form = {'family': 'serif', 'color': 'blue', 'size': 25}
        plt.bar(labels, values)
        plt.title(plot_title, fontsize= 24, color='blue')
        plt.xticks(rotation=90, fontsize=18)
        plt.yticks(fontsize=18)
        plt.xlabel(xaxis_label, fontdict=form)
        plt.ylabel(yaxis_label, fontdict=form)
        if lcount >=8:
            rotation='vertical'
        else:
            rotation='horizontal'
        for i in range(lcount):
            plt.text(i, values[i]/2, str(values[i]),fontsize=12, rotation=rotati
        plt.show()
```

Define a function to return information of the training data

- train_dir is a string specifying the full path to the directory holding the training images

The function determines the total number of training images, the class with the most image files
and their number and the class with the least image files and their number. This information is
provided to the used to enable selection of various program parameters

In [6]:
```
def check_dataset_size(train_dir):
    classes=sorted(os.listdir(train_dir))
    ftotal=0
    flargest=0
```

```
    fsmallest=100000000
    for klass in classes:
        classpath=os.path.join(train_dir,klass)
        if os.path.isdir(classpath):
            flist=os.listdir(classpath)
            fcount=len(flist)
            if fcount>flargest:
                flargest=fcount
                maxclass=klass
            if fcount < fsmallest:
                fsmallest=fcount
                minclass=klass
            ftotal += fcount
    return ftotal, flargest, maxclass, fsmallest, minclass
```

Read in data and create train, test and validation data frames

In [7]:
```
def make_dataframes(train_dir,test_dir, val_dir, limiter, min_images):
    bad_images=[]
    skip_classes=[]
    # check what directories exist
    if test_dir == None and val_dir==None:
        dirlist=[train_dir]
        names = ['train']
    elif test_dir == None:
        dirlist=[train_dir,  val_dir]
        names=['train', 'valid']
    elif val_dir == None:
        dirlist=[train_dir,  test_dir]
        names=['train', 'test']
    else:
        dirlist=[train_dir, test_dir, val_dir]
        names=['train','test', 'valid']
    ht=0 # set initial value of height counter
    wt=0  # set initial value of width counter
    total_good_files=0 # set initial value of total number of good image files c
    zipdir=zip(names, dirlist)
    for name,d in zipdir: #iterate through the  names and directories
        filepaths=[] # initialize list of filepaths
        labels=[] # initialize list of class labels
        classlist=sorted(os.listdir(d) )   # get a list of all the classes in al
        for klass in classlist: # iterate through the list of classes
            msg=f'processing images in {name} directory for class {klass}
            print(msg, '\r', end='')
            good_file_count=0 # initialize the good_file count for this class
            classpath=os.path.join(d, klass) # define the full path to the class
            if os.path.isdir(classpath): # ensure we are working with a director
                flist=sorted(os.listdir(classpath)) # make a list of all the fil
                if name == 'train' and len(flist)< min_images:
                    skip_classes.append(klass)
                else:
                    if  klass not in skip_classes:
                        if limiter != None: # check if a limiter value was speci
                            if limiter <len(flist): # if there are more files th
                                flist=np.random.choice(flist, limiter, replace=F
```

```
                    for f in flist:
                        fpath=os.path.join(classpath,f) # create the full pa
                        index=f.rfind('.')
                        ext=f[index+1:].lower() # the the file's extension
                        if ext not in ['jpg', 'jpeg', 'tiff', 'png', 'bmp']
                            bad_images.append(fpath) # if not a proper exten
                        else:
                            try: # check if image files are defective if so
                                img=cv2.imread(fpath)
                                h=img.shape[0]
                                w=img.shape[1]
                                ht +=h # add images height and width to the
                                wt += w
                                good_file_count +=1
                                total_good_files +=1
                                filepaths.append(fpath) # append the filepat
                                labels.append(klass) # append the file's cla

                            except:
                                bad_images.append(fpath) # if the image file
        print('')
        Fseries=pd.Series(filepaths, name='filepaths') # make a pandas series fo
        Lseries=pd.Series(labels, name='labels')
        df=pd.concat([Fseries, Lseries], axis=1) # make a dataframe with columns
        # depending on which directory we are iterating through create dataframe
        if name =='valid':
            valid_df=df
        elif name == 'test':
            test_df=df
        else:
            if test_dir == None and val_dir == None: # create train_df, test_df
                pdf=df
                train_df, dummy_df=train_test_split(pdf, train_size=.8, shuffle=
                valid_df, test_df=train_test_split(dummy_df, train_size=.5, shuf
            elif test_dir == None: # create just a train_df and test_df
                pdf=df
                train_df,test_df=train_test_split(pdf, train_size=.8, shuffle=Tr
            elif val_dir == None:  # create a train_df and a valid_df
                pdf=df
                train_df,valid_df=train_test_split(pdf, train_size=.8, shuffle=T
            else:
                train_df= df # test and valid dataframes exists so train_df is j
    classes=sorted(train_df['labels'].unique())
    class_count=len(classes)
    # calculate the average image height and with
    have=int(ht/total_good_files)
    wave=int(wt/total_good_files)
    aspect_ratio=have/wave
    print('number of classes in processed dataset= ', class_count)
    if len(skip_classes)>0:
        msg=f'listed below are classes not included in the dataset because it ha
        print_in_color(msg)
        for klass in skip_classes:
            print (klass)

    counts=list(train_df['labels'].value_counts())
    print('the maximum files in any class in train_df is ', max(counts), '  the
    print('train_df length: ', len(train_df), '  test_df length: ', len(test_df)
    if len(bad_images) == 0:
        print_in_color('All image files were properly processed and used in the
```

```
    else:
        print_in_color(f'the are {len(bad_images)} bad image files and {total_go
        for f in bad_images:
            print (f)
    plot_title='Images per Label in train set'
    plot_label_count (train_df,  plot_title)
    return train_df, test_df, valid_df, classes, class_count, max(counts), min(c
```

Define a function that trims the classes in a dataframe

- df is the dataframe
- max_samples is an integer specifies the maximum number of images a class can have
- min_samples is an integer specifying the minimum number of images a class must have to be included in the trimmed dataframe
- column is a string specifying the column name in the dataframe holding the class labels

In [8]:
```
def trim(df, max_samples, min_samples, column):
    # column specifies which column of the dataframe to use, typically this is t
    # df is typically train_df
    df=df.copy()
    classes=df[column].unique() # get the classes in df
    class_count=len(classes)
    length=len(df)
    print ('dataframe initially is of length ',length, ' with ', class_count, '
    groups=df.groupby(column)   # creates a set of  dataframes that only contain
    trimmed_df = pd.DataFrame(columns = df.columns) # create an empty dataframe
    for label in df[column].unique(): # iterate through each class label
        group=groups.get_group(label) # get the dataframe associate with the lab
        count=len(group) # determine how many files are in the dataframe
        if count > max_samples: # if there more files in the dataframe sample it
            sampled_group=group.sample(n=max_samples, random_state=123,axis=0)
            trimmed_df=pd.concat([trimmed_df, sampled_group], axis=0) # add the
        else:
            if count>=min_samples: # if the dataframe has more than the minimum
                sampled_group=group
                trimmed_df=pd.concat([trimmed_df, sampled_group], axis=0)
    print('after trimming, the maximum samples in any class is now ',max_samples
    classes=trimmed_df[column].unique()# return this in case some classes have l
    class_count=len(classes) # return this in case some classes have less than m
    length=len(trimmed_df)
    print ('the trimmed dataframe now is of length ',length, ' with ', class_cou
    return trimmed_df, classes, class_count
```

Expand train_df rows with augmented images so each class has n samples

- df is the dataframe
- n is an integer specifying the number of images desired for each class

- column is a string specifying the column name of the dataframe that contains the class labels
- working_dir is a string specifying the path where augmented images will be stored
- img_size is a tupple (height,width) specifying the image size of the augmented images created

In [9]:
```
def balance(df, n,column, working_dir, img_size):
    def get_augmented_image(image): # given an image this function returns an au
        width=int(image.shape[1]*.8)
        height=int(image.shape[0]*.8)
        transform= A.Compose([
            A.HorizontalFlip(p=.5),
            A.Rotate(limit=30, p=.25),
            A.RandomBrightnessContrast(p=.5),
            A.RandomGamma(p=.5),
            A.RandomCrop(width=width, height=height, p=.25) ])
        return transform(image=image)['image']
    def dummy(image):
        return image

    df=df.copy()
    print('Initial length of dataframe is ', len(df))
    aug_dir=os.path.join(working_dir, 'aug')# directory to store augmented image
    if os.path.isdir(aug_dir):# start with an empty directory
        shutil.rmtree(aug_dir)
    os.mkdir(aug_dir)
    for label in df[column].unique():
        dir_path=os.path.join(aug_dir,label)
        os.mkdir(dir_path) # make class directories within aug directory
    # create and store the augmented images
    total=0
    groups=df.groupby(column) # group by class
    for label in df[column].unique():  # for every class
        msg=f'augmenting images in train set  for class {label}
        print(msg, '\r', end='')
        group=groups.get_group(label)  # a dataframe holding only rows with the
        sample_count=len(group)   # determine how many samples there are in this
        if sample_count< n: # if the class has less than target number of images
            aug_img_count=0
            delta=n - sample_count  # number of augmented images to create
            target_dir=os.path.join(aug_dir, label)  # define where to write the
            desc=f'augmenting class {label:25s}'
            for i in range(delta):
                j= i % sample_count # need this because we may have to go throug
                img_path=group['filepaths'].iloc[j]
                img=cv2.imread(img_path)
                img=get_augmented_image(img)
                fname=os.path.basename(img_path)
                fname='aug' +str(i) +'-' +fname
                dest_path=os.path.join(target_dir, fname)
                cv2.imwrite(dest_path, img)
                aug_img_count +=1
            total +=aug_img_count
    print('')
    print('Total Augmented images created= ', total)
    # create aug_df and merge with train_df to create composite training set ndf
    aug_fpaths=[]
```

```
    aug_labels=[]
    classlist=sorted(os.listdir(aug_dir))
    for klass in classlist:
        classpath=os.path.join(aug_dir, klass)
        flist=sorted(os.listdir(classpath))
        for f in flist:
            fpath=os.path.join(classpath,f)
            aug_fpaths.append(fpath)
            aug_labels.append(klass)
    Fseries=pd.Series(aug_fpaths, name='filepaths')
    Lseries=pd.Series(aug_labels, name='labels')
    aug_df=pd.concat([Fseries, Lseries], axis=1)
    df=pd.concat([df,aug_df], axis=0).reset_index(drop=True)
    print('Length of augmented dataframe is now ', len(df))
    return df
```

Create the train_gen, test_gen and valid_gen

- batch_size is an integer specifying the generator batch size
- ycol is a string specifying the dataframe column containing the class labels
- train_df is the dataframe containing the training images
- test_df is the dataframe containing the test images
- valid_df is the dataframe containing the validation images
- image_size is a tupple (height, width)

In [10]:
```
def make_gens(batch_size, ycol, train_df, test_df, valid_df, img_size, modnum):
    if modnum == 6:
        gen=ImageDataGenerator(preprocessing_function= tf.keras.applications.inc
    else:
        gen=ImageDataGenerator()
    msg='{0:70s} for train generator'.format(' ')
    print(msg, '\r', end='') # prints over on the same line
    train_gen=gen.flow_from_dataframe(train_df, x_col='filepaths', y_col=ycol, t
                                      class_mode='categorical', color_mode='rgb
    msg='{0:70s} for valid generator'.format(' ')
    print(msg, '\r', end='') # prints over on the same line
    valid_gen=gen.flow_from_dataframe(valid_df, x_col='filepaths', y_col=ycol, t
                                      class_mode='categorical', color_mode='rgb
    # for the test_gen we want to calculate the batch size and test steps such t
    # this insures that we go through all the sample in the test set exactly onc
    length=len(test_df)
    test_batch_size=sorted([int(length/n) for n in range(1,length+1) if length %
    test_steps=int(length/test_batch_size)
    msg='{0:70s} for test generator'.format(' ')
    print(msg, '\r', end='') # prints over on the same line
    test_gen=gen.flow_from_dataframe(test_df, x_col='filepaths', y_col=ycol, tar
                                      class_mode='categorical', color_mode='rgb
    # from the generator we can get information we will need later
    classes=list(train_gen.class_indices.keys())
    class_indices=list(train_gen.class_indices.values())
    class_count=len(classes)
    labels=test_gen.labels
    return train_gen, test_gen, valid_gen, test_steps, class_count
```

Create a function to show example training images

- gen is the ImageDataGenerator containing the images to be displayed

In [11]:
```
def show_image_samples(gen, modnum ):
    msg='Below are some example training images'
    print_in_color(msg)
    t_dict=gen.class_indices
    classes=list(t_dict.keys())
    images,labels=next(gen) # get a sample batch from the generator
    plt.figure(figsize=(25, 25))
    length=len(labels)
    if length<25:   #show maximum of 25 images
        r=length
    else:
        r=25
    for i in range(r):
        plt.subplot(5, 5, i + 1)
        if modnum == 6:
            image=(images[i]+1)/2
        else:
            image=images[i] /255
        plt.imshow(image)
        index=np.argmax(labels[i])
        class_name=classes[index]
```

```
        plt.title(class_name, color='blue', fontsize=18)
        plt.axis('off')
    plt.show()
```

Create a function to calculate the F1 score metric

- y_true is an np array containing the true integer index of the associated image file
- y_pred is an np array containing the predicted integer index of a test image in the test dataframe

In [12]:
```
def F1_score(y_true, y_pred): #taken from old keras source code
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    recall = true_positives / (possible_positives + K.epsilon())
    f1_val = 2*(precision*recall)/(precision+recall+K.epsilon())
    return f1_val
```

Create a model using transfer learning

- img_size is a tupple (height, width)
- class_count is an integer that specifies the number of classes in the dataset
- lr is a float specifying the intial model's learning rate
- ans is a string. If ans is an empty string an EfficientNetB0 model is used for transfer learning,
  is ans='S' a MobilenetV3-small model is used for transfer learning. If ans='L' an EfficientNetB3 model is used for transfer learning

In [13]:
```
def make_model(img_size,class_count, lr, modnum):
    img_shape=(img_size[0], img_size[1], 3)
    if modnum ==1:
        base_model=tf.keras.applications.MobileNetV3Small(include_top=False, wei
        msg= 'created MobileNet V3  small model'
    elif modnum == 2:
        base_model=tf.keras.applications.MobileNetV3Large(include_top=False, wei
        msg='created MobileNetV3 large model'
    elif modnum == 3:
        base_model=tf.keras.applications.EfficientNetV2B0(include_top=False, wei
        msg='Created EfficientNetV2 B0 model'
    elif modnum == 4:
        base_model=tf.keras.applications.EfficientNetV2B1(include_top=False, wei
        msg='Created EfficientNetV2 B1 model'
    elif modnum == 5:
        base_model=tf.keras.applications.EfficientNetV2B2(include_top=False, wei
        msg='Created EfficientNetV2 B2 model'
    elif modnum == 6:
        tf.keras.applications.inception_resnet_v2.preprocess_input
```

```
        base_model=tf.keras.applications.InceptionResNetV2(include_top=False, we
        msg='Created InceptionResNetV2 model'
    base_model.trainable=True
    x=base_model.output
    x=BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001 )(x)
    x = Dense(256, kernel_regularizer = regularizers.l2(l = 0.016),activity_regu
                    bias_regularizer=regularizers.l1(0.006) ,activation='relu')(
    x=Dropout(rate=.4, seed=123)(x)
    output=Dense(class_count, activation='softmax')(x)
    model=Model(inputs=base_model.input, outputs=output)
    model.compile(Adamax(learning_rate=lr), loss='categorical_crossentropy', met
    msg=msg + f' with initial learning rate set to {lr}'
    print_in_color(msg)
    return model
```

Create a custom Keras callback to continue or halt training

The LR_ASK callback is a convenient callback that allows you to continue training for ask_epoch more epochs or to halt training.
If you elect to continue training for more epochs you are given the option to retain the current learning rate (LR) or to
enter a new value for the learning rate. The form of use is:
ask=LR_ASK(model,epochs, ask_epoch) where:

- model is a string which is the name of your compiled model
- epochs is an integer which is the number of epochs to run specified in model.fit
- ask_epoch is an integer. If ask_epoch is set to a value say 5 then the model will train for 5 epochs.
  then the user is ask to enter H to halt training, or enter an inter value. For example if you enter 4
  training will continue for 4 more epochs to epoch 9 then you will be queried again. Once you enter an
  integer value you are prompted to press ENTER to continue training using the current learning rate
  or to enter a new value for the learning rate.
- dwell is a boolean. If set to true the function compares the validation loss for the current tp the lowest
  validation loss thus far achieved. If the validation loss for the current epoch is larger then learning rate
  is automatically adjust by the formulanew_lr=lr * factor where factor is a float between 0 and 1. The motivation
  here is that if the validation loss increased we have moved to a point in Nspace on the cost functiob surface that
  if less favorable(higher cost) than for the epoch with the lowest cost. So the

model is loaded with the weights from the epoch with the lowest loss and the learning rate is reduced

At the end of training the model weights are set to the weights for the epoch that achieved the lowest validation loss

In [14]:
```
class LR_ASK(keras.callbacks.Callback):
    def __init__ (self, model, epochs,  ask_epoch, batches, dwell=True, factor=.
        super(LR_ASK, self).__init__()
        self.model=model
        self.ask_epoch=ask_epoch
        self.epochs=epochs
        self.ask=True # if True query the user on a specified epoch
        self.lowest_vloss=np.inf
        self.lowest_aloss=np.inf
        self.best_weights=self.model.get_weights() # set best weights to model's
        self.best_epoch=1
        self.dwell= dwell
        self.factor=factor
        self.header=True
        self.batches=batches

    def on_train_begin(self, logs=None): # this runs on the beginning of trainin
        msg1 =f'Training will proceed until epoch {self.ask_epoch} then you will
        msg2='enter H to halt training or enter an integer for how many more epo
        print_in_color(msg1 + msg2)
        if self.dwell:
            msg='learning rate will be automatically adjusted during training'
            print_in_color(msg, (0,255,0))
        self.start_time= time.time() # set the time at which training started

    def on_train_end(self, logs=None):   # runs at the end of training
        msg=f'loading model with weights from epoch {self.best_epoch}'
        print_in_color(msg, (0,255,255))
        self.model.set_weights(self.best_weights) # set the weights of the model
        tr_duration=time.time() - self.start_time   # determine how long the tra
        hours = tr_duration // 3600
        minutes = (tr_duration - (hours * 3600)) // 60
        seconds = tr_duration - ((hours * 3600) + (minutes * 60))
        msg = f'training elapsed time was {str(hours)} hours, {minutes:4.1f} min
        print_in_color (msg) # print out training duration time

    def on_epoch_begin(self, epoch, logs= None):
        self.ep_start = time.time()
    def on_train_batch_end(self, batch, logs= None):
        # get batch accuracy and loss
        acc = logs.get('accuracy') * 100
        loss = logs.get('loss')
        # prints over on the same line to show running batch count
        msg = '{0:20s}processing batch {1:} of {2:5s}- accuracy=  {3:5.3f} - los
        print(msg, '\r', end= '')

    def on_epoch_end(self, epoch, logs=None):  # method runs on the end of each
        if self.header == True:
            msg = '{0:^7s}{1:^9s}{2:^9s}{3:^9s}{4:^10s}{5:^13s}{6:^10s}{7:^13s}{
            msg1=msg.format('Epoch', 'Train', 'Train', 'Valid', 'Valid','V_Loss
            msg='{0:^7s}{1:^9s}{2:^9s}{3:^9s}{4:^10s}{5:^13s}{6:^10s}{7:^13s}{8:
            msg2=msg.format(' ', 'Loss', 'Accuracy', 'Loss', 'Accuracy','Improve
```

```
            print_in_color (msg1 + msg2)
            self.header=False
        ep_end = time.time()
        duration = ep_end - self.ep_start
        vloss=logs.get('val_loss')  # get the validation loss for this epoch
        aloss=logs.get('loss')
        acc = logs.get('accuracy')  # get training accuracy
        v_acc = logs.get('val_accuracy')  # get validation accuracy
        lr = float(tf.keras.backend.get_value(self.model.optimizer.lr)) # get th
        if epoch >0:
            deltav = self.lowest_vloss- vloss
            pimprov=(deltav/self.lowest_vloss) * 100
            deltaa=self.lowest_aloss-aloss
            aimprov=(deltaa/self.lowest_aloss) * 100
        else:
            pimprov=0.0
        if vloss< self.lowest_vloss:
            self.lowest_vloss=vloss
            self.best_weights=self.model.get_weights() # set best weights to moa
            self.best_epoch=epoch + 1
            new_lr=lr
            msg = '{0:^7s}{1:^9.4f}{2:^9.2f}{3:^9.4f}{4:^10.2f}{5:^13.2f}{6:^10.
            msg=msg.format(str(epoch+1), aloss, acc*100, vloss, v_acc*100, pimpr
            print_in_color(msg, (0,255,0)) # green foreground
        else: # validation loss increased
            if self.dwell: # if dwell is True when the validation loss increases
                lr=float(tf.keras.backend.get_value(self.model.optimizer.lr)) #
                new_lr=lr * self.factor
                msg = '{0:^7s}{1:^9.4f}{2:^9.2f}{3:^9.4f}{4:^10.2f}{5:^13.2f}{6:
                msg=msg.format(str(epoch+1), aloss, acc*100, vloss, v_acc*100, p
                print_in_color(msg, (255,255,0))
                tf.keras.backend.set_value(self.model.optimizer.lr, new_lr) # se
                self.model.set_weights(self.best_weights) # set the weights of t
        if self.ask: # are the conditions right to query the user?
            if epoch + 1 ==self.ask_epoch: # is this epoch the one for quering t
                msg='\n Enter H to end training or  an integer for the number of
                print_in_color(msg) # cyan foreground
                ans=input()
                if ans == 'H' or ans =='h' or ans == '0': # quit training for th
                    msg=f'you entered {ans},  Training halted on epoch {epoch+1}
                    print_in_color(msg)
                    self.model.stop_training = True # halt training
                else: # user wants to continue training
                    self.header=True
                    self.ask_epoch += int(ans)
                    msg=f'you entered {ans} Training will continue to epoch {sel
                    print_in_color(msg) # cyan foreground
                    if self.dwell==False:
                        lr=float(tf.keras.backend.get_value(self.model.optimizer
                        msg=f'current LR is  {lr:8.6f}  hit enter to keep  this
                        print_in_color(msg) # cyan foreground
                        ans=input(' ')
                        if ans =='':
                            msg=f'keeping current LR of {lr:7.5f}'
                            print_in_color(msg) # cyan foreground
                        else:
                            new_lr=float(ans)
                            tf.keras.backend.set_value(self.model.optimizer.lr,
                            msg=f' changing LR to {ans}'
                            print_in_color(msg) # cyan foreground
```

Define a function to plot the training data

- tr_data is the history data from model.fit

In [15]:
```
def tr_plot(tr_data):
    start_epoch=0
    #Plot the training and validation data
    tacc=tr_data.history['accuracy']
    tloss=tr_data.history['loss']
    vacc=tr_data.history['val_accuracy']
    vloss=tr_data.history['val_loss']
    tf1=tr_data.history['F1_score']
    vf1=tr_data.history['val_F1_score']
    Epoch_count=len(tacc)+ start_epoch
    Epochs=[]
    for i in range (start_epoch ,Epoch_count):
        Epochs.append(i+1)
    index_loss=np.argmin(vloss)#  this is the epoch with the lowest validation l
    val_lowest=vloss[index_loss]
    index_acc=np.argmax(vacc)
    acc_highest=vacc[index_acc]
    indexf1=np.argmax(vf1)
    vf1_highest=vf1[indexf1]
    plt.style.use('fivethirtyeight')
    sc_label='best epoch= '+ str(index_loss+1 +start_epoch)
    vc_label='best epoch= '+ str(index_acc + 1+ start_epoch)
    f1_label='best epoch= '+ str(index_acc + 1+ start_epoch)
    fig,axes=plt.subplots(nrows=1, ncols=3, figsize=(25,10))
    axes[0].plot(Epochs,tloss, 'r', label='Training loss')
    axes[0].plot(Epochs,vloss,'g',label='Validation loss' )
    axes[0].scatter(index_loss+1 +start_epoch,val_lowest, s=150, c= 'blue', labe
    axes[0].scatter(Epochs, tloss, s=100, c='red')
    axes[0].set_title('Training and Validation Loss')
    axes[0].set_xlabel('Epochs', fontsize=18)
    axes[0].set_ylabel('Loss', fontsize=18)
    axes[0].legend()
    axes[1].plot (Epochs,tacc,'r',label= 'Training Accuracy')
    axes[1].scatter(Epochs, tacc, s=100, c='red')
    axes[1].plot (Epochs,vacc,'g',label= 'Validation Accuracy')
    axes[1].scatter(index_acc+1 +start_epoch,acc_highest, s=150, c= 'blue', labe
    axes[1].set_title('Training and Validation Accuracy')
    axes[1].set_xlabel('Epochs', fontsize=18)
    axes[1].set_ylabel('Accuracy', fontsize=18)
    axes[1].legend()
    axes[2].plot (Epochs,tf1,'r',label= 'Training F1 score')
    axes[2].plot (Epochs,vf1,'g',label= 'Validation F1 score')
    index_tf1=np.argmax(tf1)#  this is the epoch with the highest training F1 sc
    tf1max=tf1[index_tf1]
    index_vf1=np.argmax(vf1)# thisiis the epoch with the highest validation F1 s
    vf1max=vf1[index_vf1]
    axes[2].scatter(index_vf1+1 +start_epoch,vf1max, s=150, c= 'blue', label=vc_
    axes[2].scatter(Epochs, tf1, s=100, c='red')
    axes[2].set_title('Training and Validation F1 score')
    axes[2].set_xlabel('Epochs', fontsize=18)
```

```
    axes[2].set_ylabel('F1  score', fontsize=18)
    axes[2].legend()
    plt.tight_layout
    plt.show()
    return
```

Define a function to save the training data to a csv file

- history is the history from history=model.fit
- csvpath is a string specifying the full path where the csv file will be stored

In [16]:
```
def save_history_to_csv(history,csvpath):
    trdict=history.history
    df=pd.DataFrame()
    df['Epoch']=list(np.arange(1, len(trdict['loss']) + 1 ))
    keys=list(trdict.keys())
    for key in keys:
        data=list(trdict[key])
        df[key]=data
    df.to_csv(csvpath, index=False)
```

Make Predictions on the test set

Define a function which takes in a trained model and a test generator and generates predictions

on the test set including a confusion matrix and a classification report

- model is the trained model
- test_gen is the ImageDataGenerator holding the test filepaths and labels

The function will produce a classification report, a confusion matrix and a plot of misclassifications

In [17]:
```
def predictor(model,test_gen):
    classes=list(test_gen.class_indices.keys())
    class_count=len(classes)
    preds=model.predict(test_gen, verbose=1)
    errors=0
    test_count =len(preds)
    misclassified_classes=[]
    misclassified_files=[]
    misclassified_as = []
    pred_indices=[]
    for i, p in enumerate (preds):
        pred_index=np.argmax(p)
        pred_indices.append(pred_index)
        true_index= test_gen.labels[i]
```

```
        if  pred_index != true_index:
            errors +=1
            misclassified_classes.append(classes[true_index])
            misclassified_as.append(classes[pred_index])
            file=test_gen.filenames[i]
            split=file.split('/')
            L=len(split)
            f=split[L-2] +' '+ split[L-1]
            misclassified_files.append(f)

    accuracy = (test_count-errors)*100/test_count
    ytrue=np.array(test_gen.labels)
    ypred=np.array(pred_indices)
    f1score=f1_score(ytrue, ypred, average='weighted')* 100
    msg=f'There were {errors} errors in {test_count} tests for an accuracy of {a
    print (msg)
    misclassified_classes=sorted(misclassified_classes)
    if len(misclassified_classes) > 0:
        misclassifications=[]
        for klass in misclassified_classes:
            mis_count=misclassified_classes.count(klass)
            misclassifications.append(mis_count)
        unique=len(np.unique(misclassified_classes))
        if unique==1:
            height=int(unique)
        else:
            height =int(unique/2)
        plt.figure(figsize=(10, height))
        plt.style.use('fivethirtyeight')
        plt.barh(misclassified_classes, misclassifications )
        plt.title( 'Classification Errors on Test Set by Class', fontsize=20, co
        plt.xlabel('NUMBER OF MISCLASSIFICATIONS', fontsize=20, color='blue')
        plt.ylabel('CLASS', fontsize=20, color='blue')
        plt.show()
    if class_count <=30:
        cm = confusion_matrix(ytrue, ypred )
        # plot the confusion matrix
        plt.figure(figsize=(12, 8))
        sns.heatmap(cm, annot=True, vmin=0, fmt='g', cmap='Blues', cbar=False)
        plt.xticks(np.arange(class_count)+.5, classes, rotation=90)
        plt.yticks(np.arange(class_count)+.5, classes, rotation=0)
        plt.xlabel("Predicted")
        plt.ylabel("Actual")
        plt.title("Confusion Matrix")
        plt.show()
        clr = classification_report(ytrue, ypred, target_names=classes, digits=
        print("Classification Report:\n----------------------\n", clr)
    return f1score, misclassified_files
```

Save the model

In [18]:
```
def save_model(model,subject, classes, img_size, f1score, working_dir):
    name=f'{subject}-{str(len(classes))}-({str(img_size[0])} X {str(img_size[1])
    model_save_loc=os.path.join(working_dir, name)
    try:
        model.save(model_save_loc)
        msg= f'model was saved as {model_save_loc}'
```

```
        print_in_color(msg, (0,255,255),(0,0,0))
    except:
        msg='model can not be saved due to tensorflow 2.10.0 or higher. Bug invo
        print_in_color(msg, (0,255,255), (0,0,0)) # cyan foreground
```

Define a function that runs the classifier

This is the code that provides the interface to the user and calls the various defined functions based on the users input. To follow how the program works start here. You can run this function as many times as you care to if you want to do runs with various different parameters. You can save the data from each run in a csv file to compare training and validation results from different runs.

In [19]:
```
def run(working_dir= None,train_dir=None, test_dir=None, valid_dir=None,start=Tr
    if start:
        msg='enter the full path to the working directory where data will be sto
        print_in_color(msg)
        working_dir=input(' ')
        if 'kaggle' in working_dir: # if running the notebook on kaggle need to
            delimiter='/'
        else:
            delimiter='\\'
        msg='Enter the full path to the train directory'
        print_in_color(msg)
        train_dir =input(' ')
        msg = 'Enter the full path to the validation directory. If there is no v
        print_in_color(msg)
        ans=input(' ')
        if ans == '':
            valid_dir=None
        else:
            valid_dir=ans
        msg = 'Enter the full path to the test directory. If there is no test di
        print_in_color(msg)
        ans=input(' ')
        if ans == '':
            test_dir=None
        else:
            test_dir=ans

    ftotal, flargest, maxclass , fsmallest, minclass= check_dataset_size(train_d
    msg1= f'the train directory contains {ftotal} files, class {maxclass} has th
    msg2= f'class {minclass} contains the least number of images of {fsmallest}
    msg3= f'NOTE if the value of most images is the same as the value of least i
    msg4='When dealing with very large data sets to save time you may not want t
    msg5 ='to limit the maximum number of image files in any class you can enter
    msg=msg1 +msg2 + msg3 + msg4 + msg5
    print_in_color(msg) # print the training data information for the user
    msg = 'input a limiter integer value to limit max number of images in a clas
    print_in_color(msg)
    limiter=input(' ')
    if limiter == '':
        limiter = None
```

```
        msg='dataset will be processed as is with no limiter'
        print_in_color(msg)
    else:
        limiter=int(limiter)
        msg=f'images will be limited to a maximum of {limiter} images in each cl
        print_in_color(msg)
    msg='enter the minimum number of training images a class must have to be inc
    print_in_color(msg)
    min_images=int(input(' '))
    if min_images=='':
        min_images=10
    else:
        min_images=int(min_images)
    # create train, test and valid data frames
    train_df, test_df, valid_df, classes, class_count, max_samples, min_samples,
    msg=f'the average height of training images is {have} and average width is {
    print_in_color(msg)
    img_height=int(input(' '))
    msg=' Enter the image width to be used to train the model'
    print_in_color(msg)
    img_width=int(input(' '))
    img_size=(img_height, img_width)
    msg=f'model will be trained with image shape of  ( {img_height}, {img_width}
    print_in_color(msg)
    # determine if user wants to balance the training dataframe
    msg=' enter A  to auto balance the train set or enter or hit enter to leave
    print_in_color(msg)
    ans=input(' ')
    if ans == 'A' or ans =='a':
        msg='enter the number of images you want to have in each class of the tr
        print_in_color(msg)
        max_images=int(input(' '))
        train_df, classes, class_count=trim (train_df, max_images, min_images, '
        train_df=balance(train_df, max_images, 'labels', working_dir, img_size)
        plot_title='Images per Label after Auto Balance of train data set'
        plot_label_count (train_df,  plot_title)  # plot the number of images in
    else:
        msg=f'training data set will be used as is '
        print_in_color (msg)
    classes=list(train_df['labels'].unique())
    class_count = len(classes)
    msg='enter the desired batch size or press enter to use the default batch si
    print_in_color(msg)
    ans=input(' ')
    if ans =='':
        bs=20
    else:
        bs=int(ans)
    # make the train, test and valid data generators
    print ('select a model number from the list below ',flush=True)
    print ('{0:^9s}{1:^20s}{2:^12s}'.format('Model No', 'Model Type', 'Parameter
    models=['MobileNetV3-small', 'MobileNetV3-large', 'EfficientNetV2B0', 'Effic
    parameters=[' 1.0 M', '3.3 M', '6.3 M', '7.3 M','9.1 M', '54.7 M']
    for i in range(0,6):
        msg='{0:^9s}{1:^20s}{2:^12s}'.format(str(i+1),models[i], parameters[i])
        print (msg)
    modnum=int(input(" "))
    lr=.001
    model=make_model(img_size,class_count, lr, modnum)
    msg1='enter the number of epochs to run initially. After these epochs comple
```

```
    msg2='or enter an integer for how many more epochs to run then be asked agai
    print(msg1 + msg2)
    ask_epoch=int(input(' '))
    batches=int(len(train_df)/bs)
    # instantiate the custom callback
    epochs=100
    ask=LR_ASK(model, epochs=epochs,  ask_epoch=ask_epoch, batches=batches) # in
    callbacks=[ask]
    # make the train, test and valid data generators
    train_gen, test_gen, valid_gen, test_steps, class_count= make_gens(bs, 'labe
    show_image_samples(train_gen, modnum ) # show some sample training images
    # train the model
    history=model.fit(x=train_gen,   epochs=epochs, verbose=0, callbacks=callbac
                   validation_steps=None,  shuffle=True,  initial_epoch=0) # tra

    tr_plot(history) # plot training data
    msg='To save the training data to a csv file enter the name for the csv file
    print_in_color(msg)
    ans=input(' ')
    if ans !='': # save the training data to a csv file history.csv in working d
        csvpath=os.path.join(working_dir, ans + '.csv')
        save_history_to_csv(history,csvpath)
        msg=f'training data saved to {csvpath}'
        print_in_color(msg)
    # make predictions on the test set, create classification report and confusi
    f1score, misclassified_files=predictor(model,test_gen) # do predictions on t
    if len(misclassified_files)>0:
        print ('below is a list of misclassified test files:')
        for f in misclassified_files:
            print (f)
    # save the model
    msg=f'your trained model will be saved to directory {working_dir} enter a su
    print_in_color(msg)
    subject=input(' ')
    save_model(model,subject, classes, img_size, f1score, working_dir) # save th
    msg='model save nomenclature is directory/subject-number of classes- (img_he
    print_in_color(msg)
    # remove the aug directory from the working directory
    if os.path.isdir(os.path.join(working_dir, 'aug')):
        shutil.rmtree(os.path.join(working_dir, 'aug')) #remove the augmentation
    return working_dir,train_dir, test_dir, valid_dir, model
```

Initiate the run function and enable ability to execute run multiple times

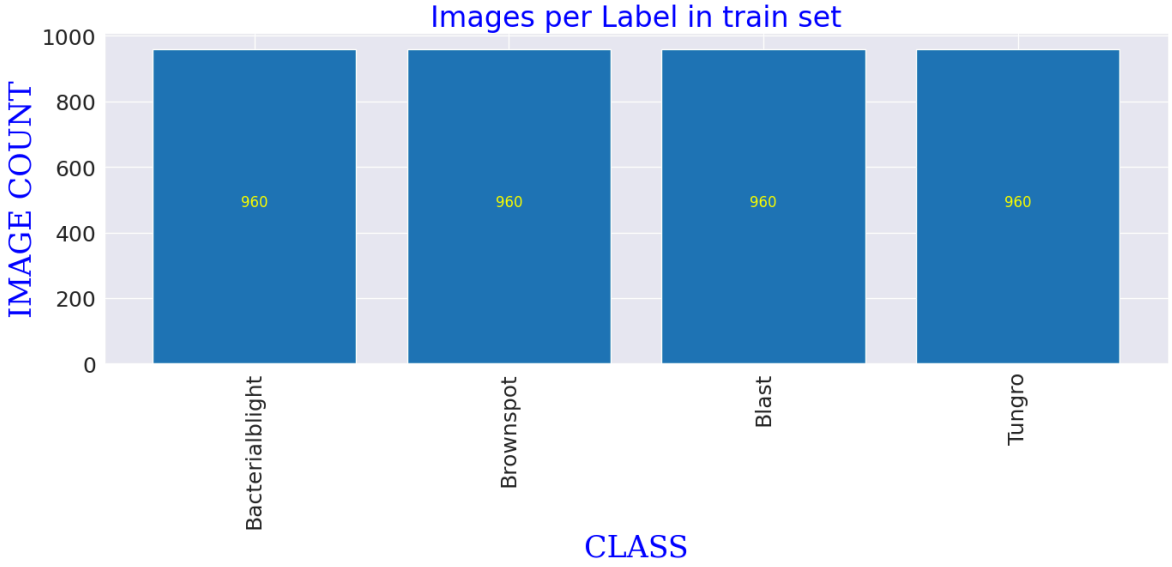
The code below initiate the first run of the classifier and lets the user rerun the classifier

Save the model

In [20]:
```
working_dir,train_dir, test_dir, valid_dir, model=run(start=True) # run the clas
stop = False
while stop == False:
    msg='enter R to rerun the classifier or press enter to quit  '
```

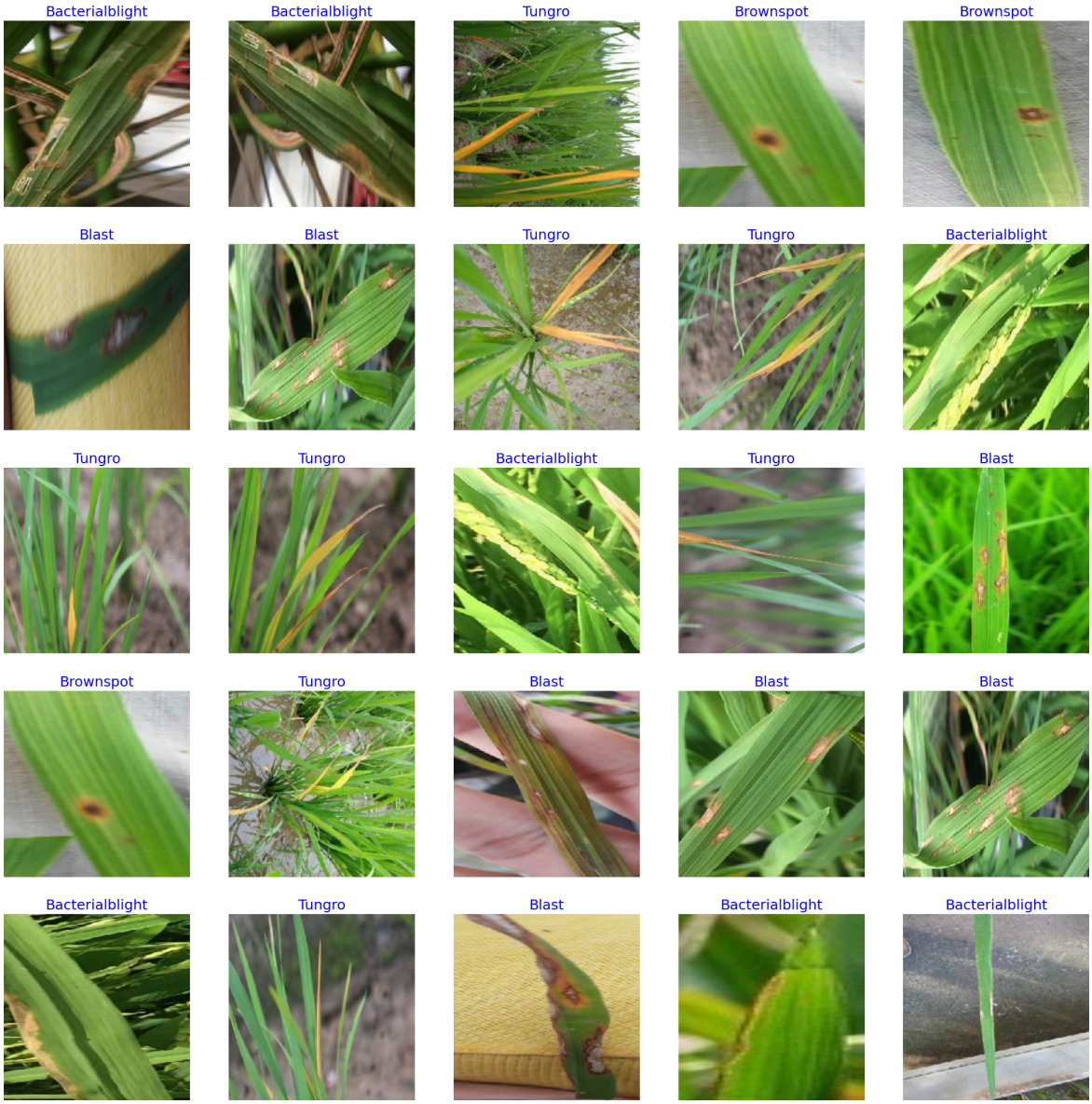
```
    print_in_color(msg)
    ans=input(' ')
    if ans == 'R' or ans == 'r':
        msg='Enter N to specify new data directories or hit Enter to use existin
        print_in_color(msg)
        ans=input(' ')
        if ans == '':
            run(working_dir,train_dir=train_dir, test_dir=test_dir, valid_dir=va
        else:
            run(True)
    else:
        stop = True
        print_in_color ('process commpleted')
```

```
enter the full path to the working directory where data will be stored.
Enter the full path to the train directory
Enter the full path to the validation directory. If there is no validation direct
ory press enter
Enter the full path to the test directory. If there is no test directory press en
ter
the train directory contains 5932 files, class Brownspot has the most images of 1
600 files
class Tungro contains the least number of images of 1308 files
NOTE if the value of most images is the same as the value of least images the dat
aset is balanced
When dealing with very large data sets to save time you may not want to read in a
ll the image files
to limit the maximum number of image files in any class you can enter an integer
limiter value
input a limiter integer value to limit max number of images in a class, for no li
mit hit enter
images will be limited to a maximum of 1200 images in each class
enter the minimum number of training images a class must have to be included in t
he dataset or press enter to use the default value of 10
processing images in train directory for class Tungro
number of classes in processed dataset=  4
the maximum files in any class in train_df is  960   the minimum files in any cla
ss in train_df is  960
train_df length:  3840   test_df length:  480   valid_df length:  480
All image files were properly processed and used in the dataframes
```




```
the average height of training images is 313 and average width is 314
  enter the image height to be used to train the model
 Enter the image width to be used to train the model
```

```
model will be trained with image shape of  ( 224, 224 )
 enter A  to auto balance the train set or enter or hit enter to leave train set
unchanged
training data set will be used as is
enter the desired batch size or press enter to use the default batch size of 20
select a model number from the list below
Model No      Model Type      Parameters
    1     MobileNetV3-small      1.0 M
    2     MobileNetV3-large     3.3 M
    3      EfficientNetV2B0     6.3 M
    4      EfficientNetV2B1     7.3 M
    5      EfficientNetV2B2     9.1 M
    6      InceptionResNetV2    54.7 M
Downloading data from https://storage.googleapis.com/tensorflow/keras-application
s/efficientnet_v2/efficientnetv2-b0_notop.h5
24274472/24274472 [==============================] - 0s 0us/step
Created EfficientNetV2 B0 model with initial learning rate set to 0.001
enter the number of epochs to run initially. After these epochs complete you can
enter H to halt training
or enter an integer for how many more epochs to run then be asked again
Found 3840 validated image filenames belonging to 4 classes.           for train
generator
Found 480 validated image filenames belonging to 4 classes.            for valid
generator
Found 480 validated image filenames belonging to 4 classes.            for test g
enerator
Below are some example training images
```



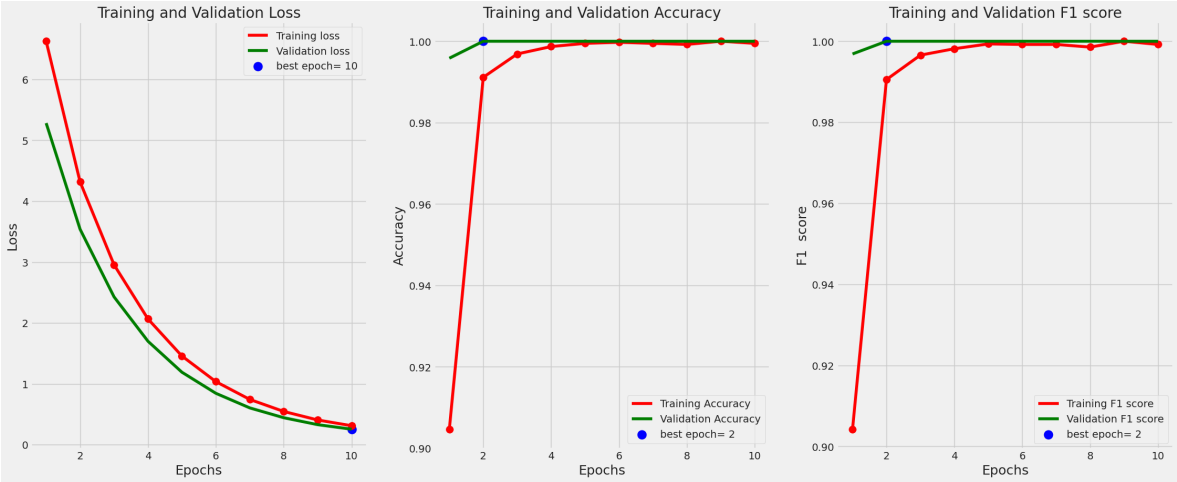

```
Training will proceed until epoch 10 then you will be asked to
enter H to halt training or enter an integer for how many more epochs to run then
be asked again
learning rate will be automatically adjusted during training
```

```
2024-01-01 12:03:41.314932: E tensorflow/core/grappler/optimizers/meta_optimizer.
cc:954] layout failed: INVALID_ARGUMENT: Size of values 0 does not match size of
permutation 4 @ fanin shape inmodel/block2b_drop/dropout/SelectV2-2-TransposeNHWC
ToNCHW-LayoutOptimizer
```

Epoch	Train Loss	Train Accuracy	Valid Loss	Valid Accuracy	V_Loss % Improvement	Learning Rate	Next LR	Duration in Seconds
1	6.6278	90.47	5.2828	99.58	0.00	0.001000	0.001000	76.01
2	4.3160	99.11	3.5387	100.00	33.01	0.001000	0.001000	23.55
3	2.9557	99.69	2.4282	100.00	31.38	0.001000	0.001000	23.75
4	2.0656	99.87	1.6991	100.00	30.03	0.001000	0.001000	23.50
5	1.4564	99.95	1.1903	100.00	29.94	0.001000	0.001000	23.33
6	1.0359	99.97	0.8439	100.00	29.10	0.001000	0.001000	23.34
7	0.7431	99.95	0.6039	100.00	28.44	0.001000	0.001000	23.41
8	0.5456	99.92	0.4427	100.00	26.70	0.001000	0.001000	23.54
9	0.4057	100.00	0.3285	100.00	25.80	0.001000	0.001000	23.46
10	0.3111	99.95	0.2542	100.00	22.61	0.001000	0.001000	23.39

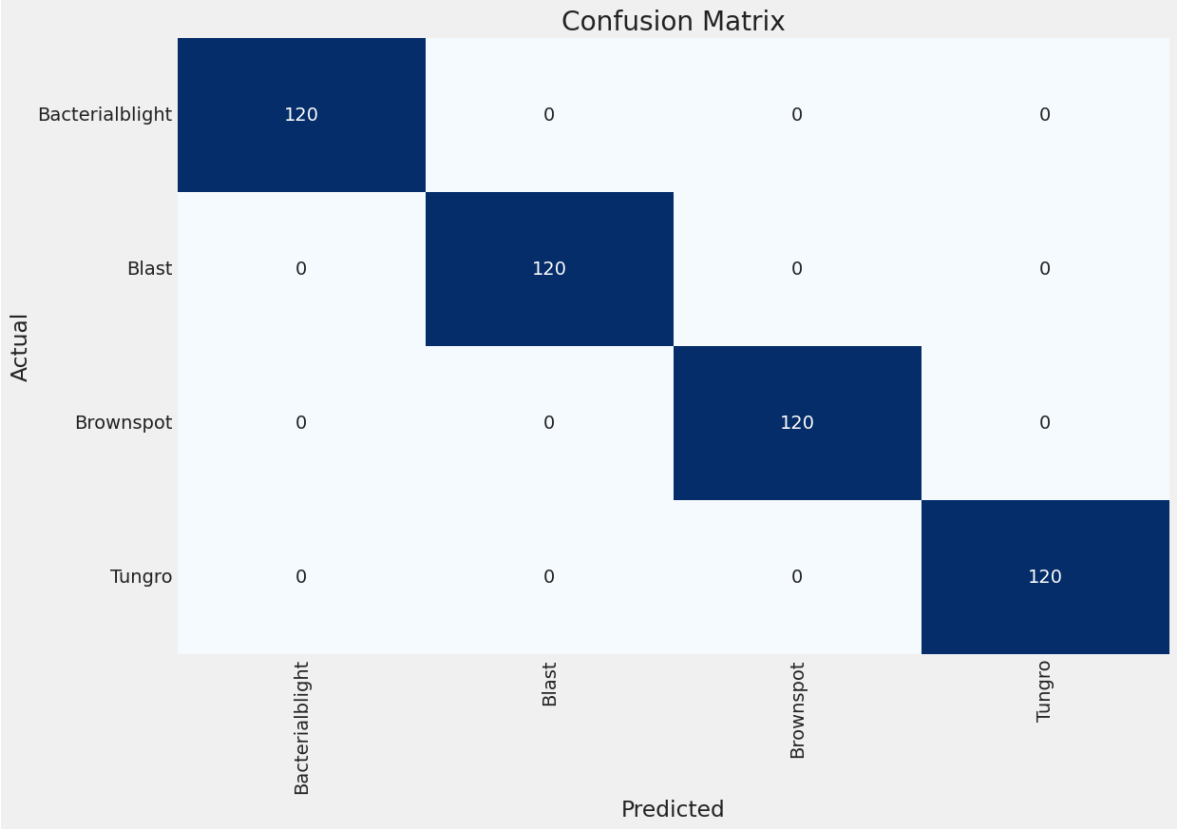
Enter H to end training or an integer for the number of additional epochs to run then ask again
you entered h, Training halted on epoch 10 due to user input

loading model with weights from epoch 10
training elapsed time was 0.0 hours, 4.0 minutes, 53.01 seconds)



To save the training data to a csv file enter the name for the csv file or press enter to not save the data

```
6/6 [==============================] - 4s 158ms/step
There were 0 errors in 480 tests for an accuracy of 100.00 and an F1 score of 100.00
```



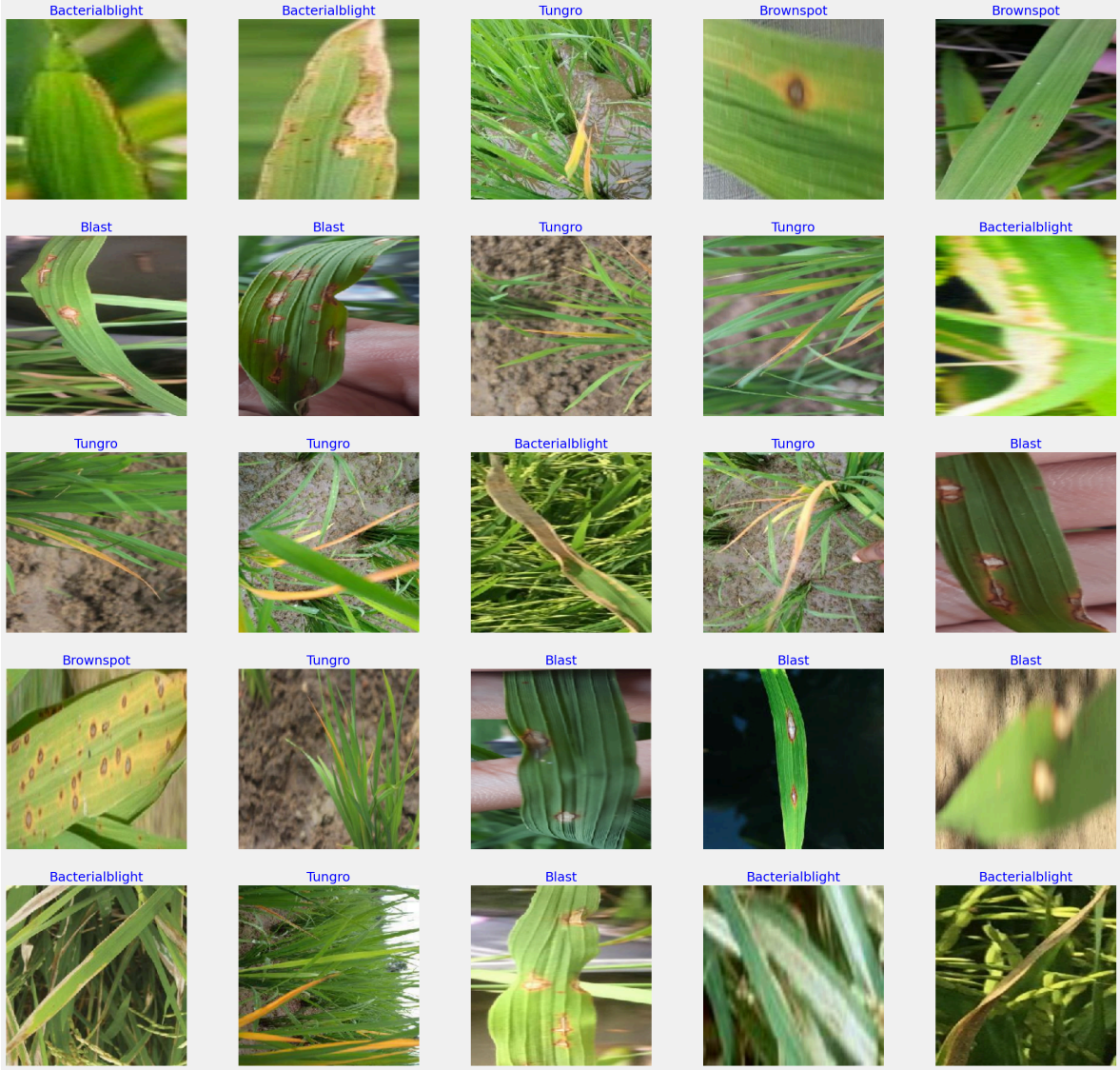
Classification Report:

	precision	recall	f1-score	support
Bacterialblight	1.0000	1.0000	1.0000	120
Blast	1.0000	1.0000	1.0000	120
Brownspot	1.0000	1.0000	1.0000	120
Tungro	1.0000	1.0000	1.0000	120
accuracy			1.0000	480
macro avg	1.0000	1.0000	1.0000	480
weighted avg	1.0000	1.0000	1.0000	480

```
your trained model will be saved to directory /kaggle/working/ enter a subject fo
r the saved model
model was saved as /kaggle/working/EfficientNetB0-4-(224 X 224)- 100.00.h5
model save nomenclature is directory/subject-number of classes- (img_height, img_
width)- F1score.h5
enter R to rerun the classifier or press enter to quit
Enter N to specify new data directories or hit Enter to use existing data directo
ry paths
the train directory contains 5932 files, class Brownspot has the most images of 1
600 files
class Tungro contains the least number of images of 1308 files
NOTE if the value of most images is the same as the value of least images the dat
aset is balanced
When dealing with very large data sets to save time you may not want to read in a
ll the image files
to limit the maximum number of image files in any class you can enter an integer
limiter value
input a limiter integer value to limit max number of images in a class, for no li
mit hit enter
images will be limited to a maximum of 1200 images in each class
enter the minimum number of training images a class must have to be included in t
he dataset or press enter to use the default value of 10
```

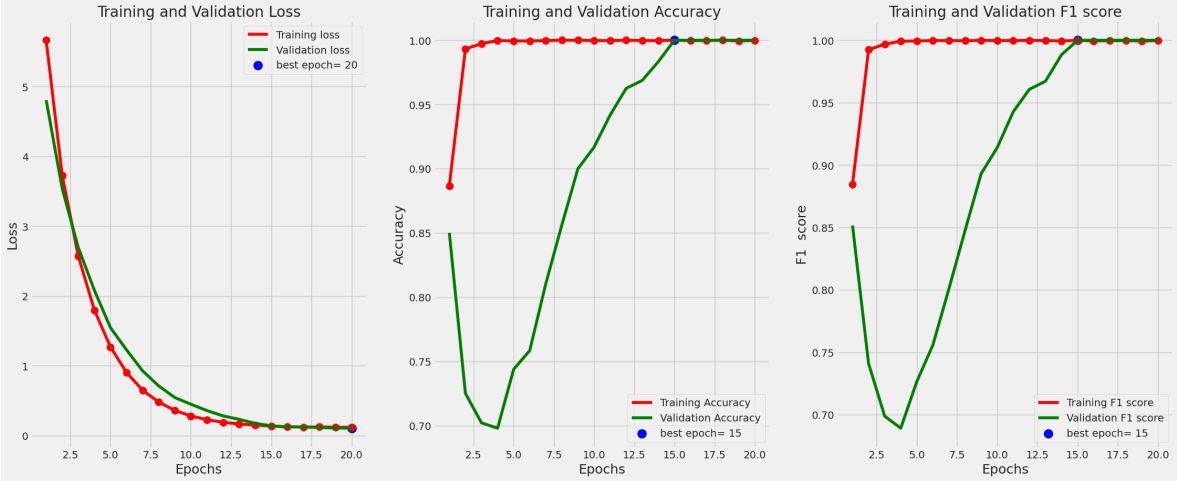
```
processing images in train directory for class Tungro
number of classes in processed dataset=  4
the maximum files in any class in train_df is  960   the minimum files in any cla
ss in train_df is  960
train_df length:  3840   test_df length:  480   valid_df length:  480
All image files were properly processed and used in the dataframes
```


```
the average height of training images is 313 and average width is 314
 enter the image height to be used to train the model
 Enter the image width to be used to train the model
model will be trained with image shape of  ( 224, 224 )
 enter A  to auto balance the train set or enter or hit enter to leave train set
unchanged
training data set will be used as is
enter the desired batch size or press enter to use the default batch size of 20
select a model number from the list below
Model No      Model Type      Parameters
    1     MobileNetV3-small      1.0 M
    2     MobileNetV3-large     3.3 M
    3      EfficientNetV2B0     6.3 M
    4      EfficientNetV2B1     7.3 M
    5      EfficientNetV2B2     9.1 M
    6      InceptionResNetV2    54.7 M
Downloading data from https://storage.googleapis.com/tensorflow/keras-application
s/mobilenet_v3/weights_mobilenet_v3_small_224_1.0_float_no_top_v2.h5
4334752/4334752 [==============================] - 0s 0us/step
created MobileNet V3  small model with initial learning rate set to 0.001
enter the number of epochs to run initially. After these epochs complete you can
enter H to halt training
or enter an integer for how many more epochs to run then be asked again
Found 3840 validated image filenames belonging to 4 classes.           for train
generator
Found 480 validated image filenames belonging to 4 classes.            for valid
generator
Found 480 validated image filenames belonging to 4 classes.            for test g
enerator
Below are some example training images
```

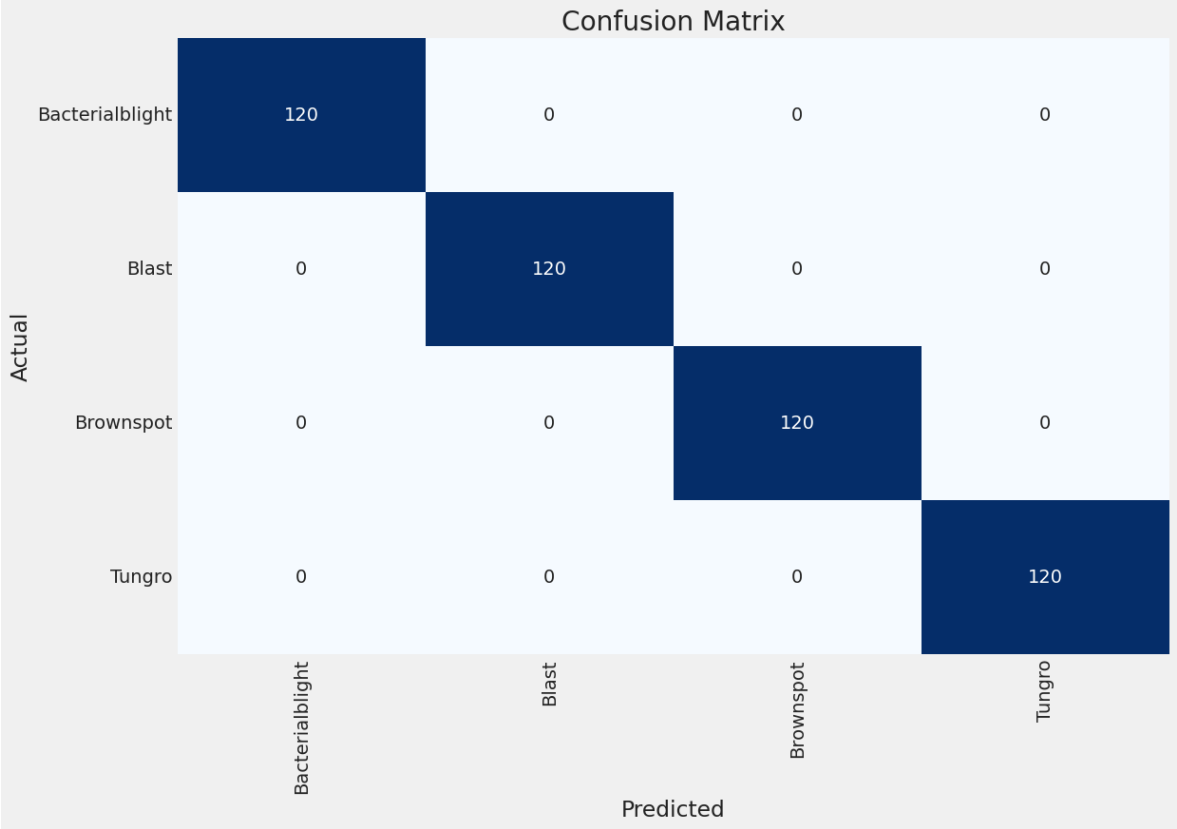

Bacterialblight
Bacterialblight
Tungro
Brownspot
Brownspot
Blast
Blast
Tungro
Tungro
Bacterialblight
Tungro
Tungro
Bacterialblight
Tungro
Blast
Brownspot
Tungro
Blast
Blast
Blast
Bacterialblight
Tungro
Blast
Bacterialblight
Bacterialblight

```
Training will proceed until epoch 10 then you will be asked to
enter H to halt training or enter an integer for how many more epochs to run then be asked again
learning rate will be automatically adjusted during training
 Epoch   Train    Train    Valid    Valid     V_Loss %    Learning    Next LR   Duration in
         Loss   Accuracy   Loss    Accuracy  Improvement    Rate        Rate      Seconds
   1    5.6640    88.67   4.8065    85.00        0.00      0.001000    0.001000    37.67
   2    3.7287    99.32   3.5380    72.50       26.39      0.001000    0.001000    11.16
   3    2.5715    99.74   2.6927    70.21       23.89      0.001000    0.001000    10.57
   4    1.7984    99.97   2.0817    69.79       22.69      0.001000    0.001000    11.80
   5    1.2703    99.95   1.5470    74.37       25.69      0.001000    0.001000    10.82
   6    0.9047    99.95   1.2278    75.83       20.63      0.001000    0.001000    10.34
   7    0.6511    99.97   0.9282    81.04       24.41      0.001000    0.001000    10.71
   8    0.4811   100.00   0.7125    85.62       23.24      0.001000    0.001000    10.45
   9    0.3627   100.00   0.5460    90.00       23.36      0.001000    0.001000    10.50
  10    0.2814    99.97   0.4500    91.67       17.59      0.001000    0.001000    10.22

 Enter H to end training or  an integer for the number of additional epochs to run then ask again
you entered 5 Training will continue to epoch 15
 Epoch   Train    Train    Valid    Valid     V_Loss %    Learning    Next LR   Duration in
         Loss   Accuracy   Loss    Accuracy  Improvement    Rate        Rate      Seconds
  11    0.2290    99.97   0.3587    94.17       20.28      0.001000    0.001000    10.52
  12    0.1913   100.00   0.2830    96.25       21.10      0.001000    0.001000    11.33
  13    0.1665    99.97   0.2335    96.88       17.49      0.001000    0.001000    11.90
  14    0.1518    99.97   0.1764    98.33       24.47      0.001000    0.001000    10.41
  15    0.1358   100.00   0.1398   100.00       20.77      0.001000    0.001000    10.92

 Enter H to end training or  an integer for the number of additional epochs to run then ask again
```

```
you entered 5 Training will continue to epoch 20
 Epoch   Train     Train     Valid     Valid      V_Loss %    Learning     Next LR    D
uration in
         Loss    Accuracy    Loss    Accuracy  Improvement     Rate         Rate
Seconds
  16    0.1271    99.97    0.1242    100.00       11.15      0.001000     0.001000
11.40
  17    0.1226    99.97    0.1252    100.00       -0.81      0.001000     0.000400
10.89
  18    0.1246   100.00    0.1178    100.00        5.16      0.000400     0.000400
11.01
  19    0.1213    99.95    0.1082    100.00        8.10      0.000400     0.000400
10.62
  20    0.1185    99.97    0.1066    100.00        1.51      0.000400     0.000400
10.84

 Enter H to end training or  an integer for the number of additional epochs to ru
n then ask again
you entered h,  Training halted on epoch 20 due to user input

loading model with weights from epoch 20
training elapsed time was 0.0 hours,  4.0 minutes, 18.96 seconds)
```




```
To save the training data to a csv file enter the name for the csv file or press
enter to not save the data
6/6 [==============================] - 3s 157ms/step
There were 0 errors in 480 tests for an accuracy of 100.00 and an F1 score of 10
0.00
```



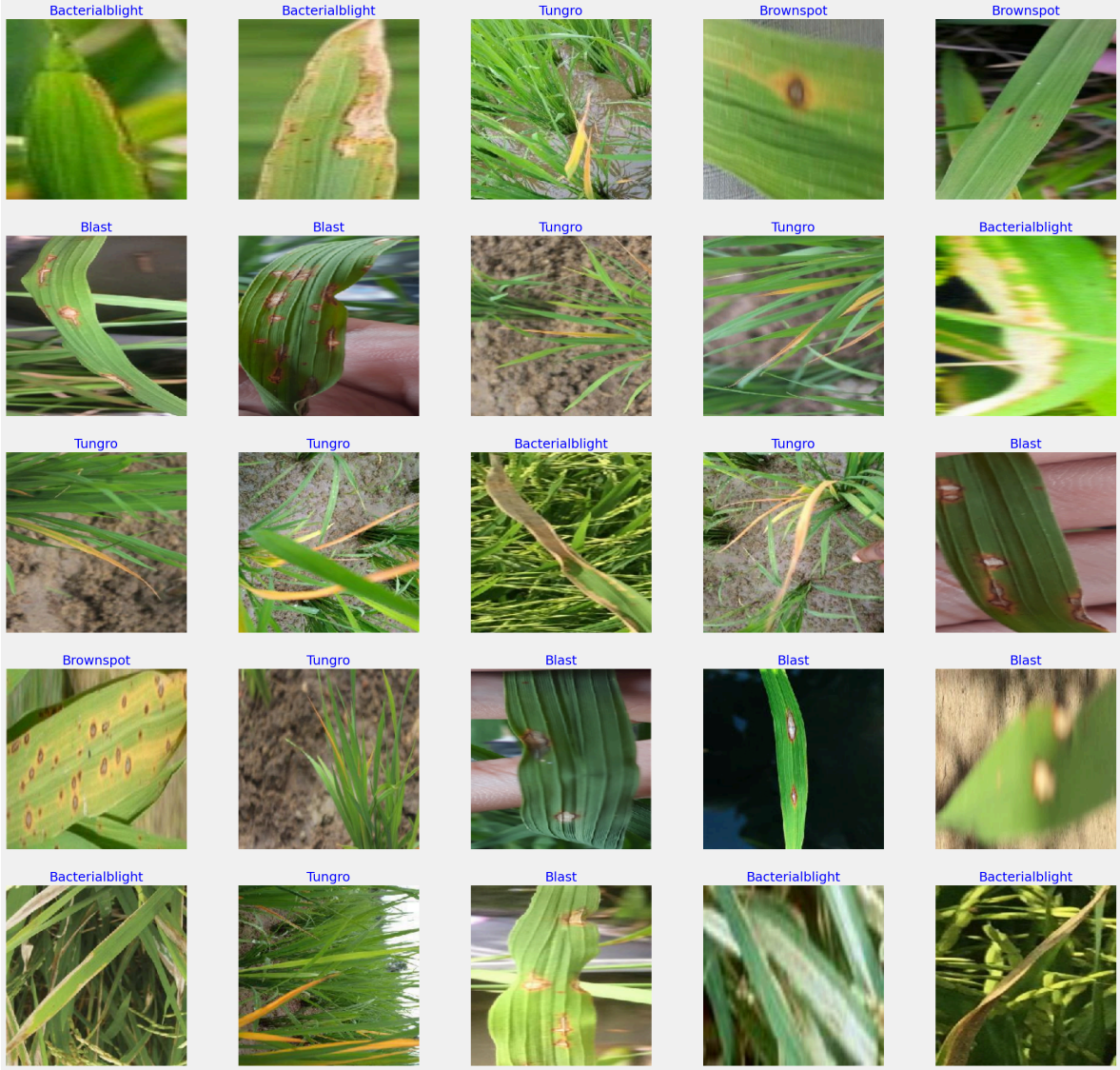
Classification Report:

	precision	recall	f1-score	support
Bacterialblight	1.0000	1.0000	1.0000	120
Blast	1.0000	1.0000	1.0000	120
Brownspot	1.0000	1.0000	1.0000	120
Tungro	1.0000	1.0000	1.0000	120
accuracy			1.0000	480
macro avg	1.0000	1.0000	1.0000	480
weighted avg	1.0000	1.0000	1.0000	480

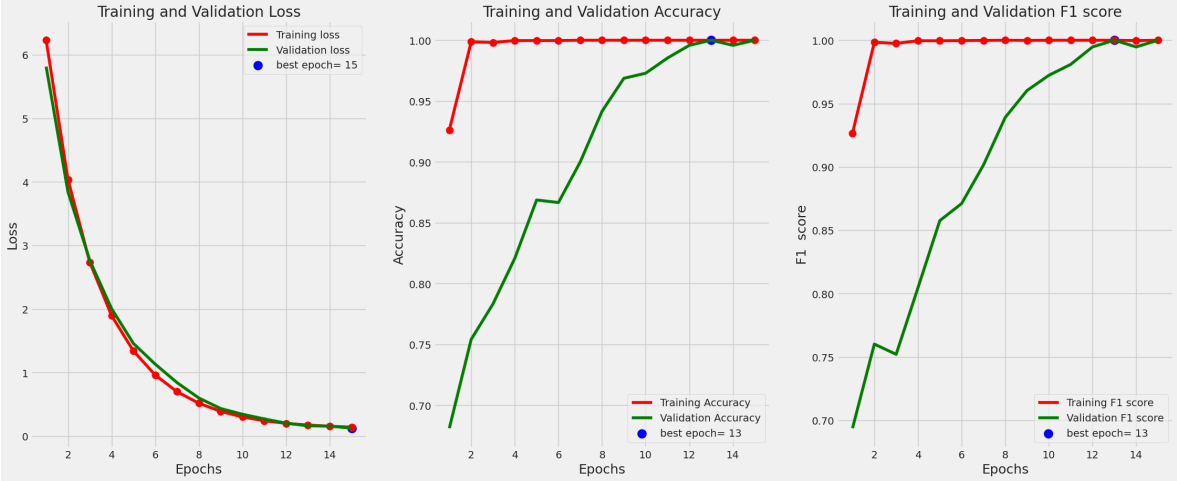
```
your trained model will be saved to directory /kaggle/working/ enter a subject fo
r the saved model
model was saved as /kaggle/working/MobileNetV3 small-4-(224 X 224)- 100.00.h5
model save nomenclature is directory/subject-number of classes- (img_height, img_
width)- F1score.h5
enter R to rerun the classifier or press enter to quit
Enter N to specify new data directories or hit Enter to use existing data directo
ry paths
the train directory contains 5932 files, class Brownspot has the most images of 1
600 files
class Tungro contains the least number of images of 1308 files
NOTE if the value of most images is the same as the value of least images the dat
aset is balanced
When dealing with very large data sets to save time you may not want to read in a
ll the image files
to limit the maximum number of image files in any class you can enter an integer
limiter value
input a limiter integer value to limit max number of images in a class, for no li
mit hit enter
images will be limited to a maximum of 1200 images in each class
enter the minimum number of training images a class must have to be included in t
he dataset or press enter to use the default value of 10
```

```
processing images in train directory for class Tungro
number of classes in processed dataset=  4
the maximum files in any class in train_df is  960   the minimum files in any cla
ss in train_df is  960
train_df length:  3840   test_df length:  480   valid_df length:  480
All image files were properly processed and used in the dataframes
```


```
the average height of training images is 313 and average width is 314
 enter the image height to be used to train the model
 Enter the image width to be used to train the model
model will be trained with image shape of  ( 224, 224 )
 enter A  to auto balance the train set or enter or hit enter to leave train set
unchanged
training data set will be used as is
enter the desired batch size or press enter to use the default batch size of 20
select a model number from the list below
Model No      Model Type      Parameters
    1     MobileNetV3-small      1.0 M
    2     MobileNetV3-large     3.3 M
    3      EfficientNetV2B0     6.3 M
    4      EfficientNetV2B1     7.3 M
    5      EfficientNetV2B2     9.1 M
    6      InceptionResNetV2    54.7 M
Downloading data from https://storage.googleapis.com/tensorflow/keras-application
s/mobilenet_v3/weights_mobilenet_v3_large_224_1.0_float_no_top_v2.h5
12683000/12683000 [==============================] - 0s 0us/step
created MobileNetV3 large model with initial learning rate set to 0.001
enter the number of epochs to run initially. After these epochs complete you can
enter H to halt training
or enter an integer for how many more epochs to run then be asked again
Found 3840 validated image filenames belonging to 4 classes.           for train
generator
Found 480 validated image filenames belonging to 4 classes.            for valid
generator
Found 480 validated image filenames belonging to 4 classes.            for test g
enerator
Below are some example training images
```

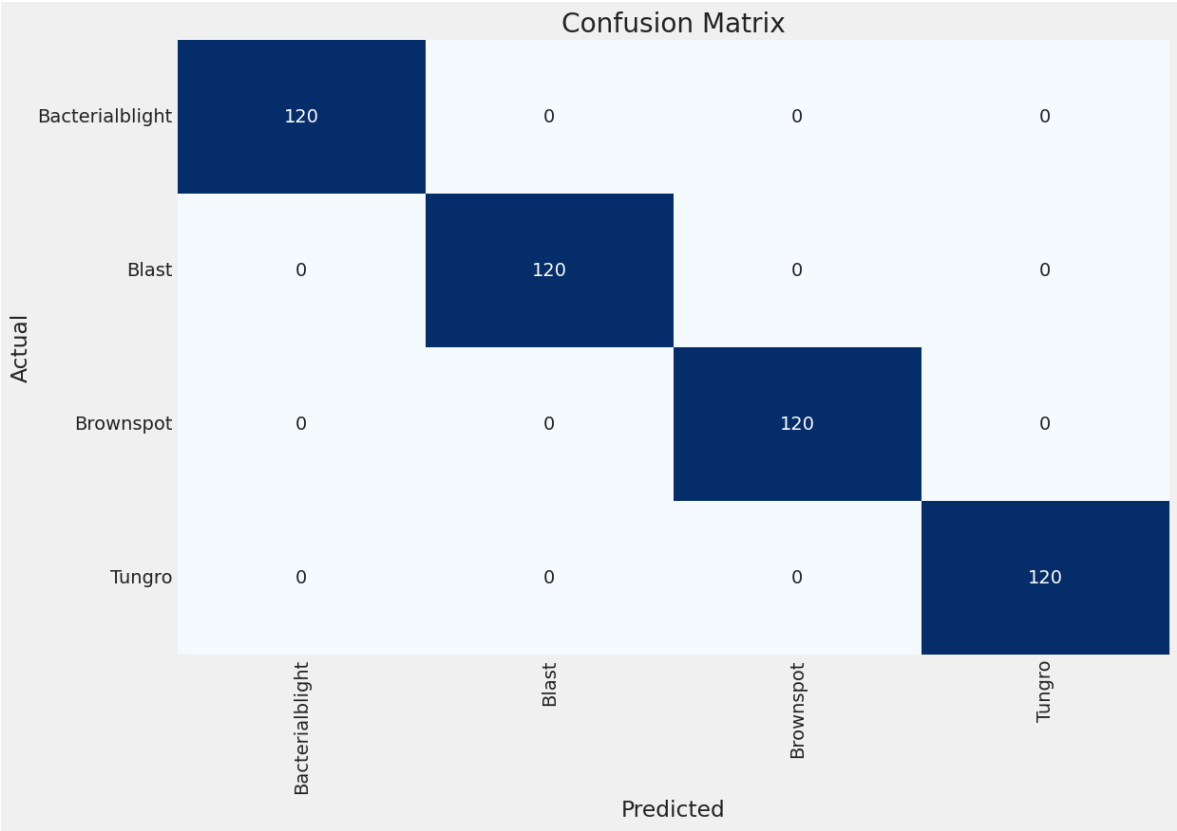

Bacterialblight
Bacterialblight
Tungro
Brownspot
Brownspot
Blast
Blast
Tungro
Tungro
Bacterialblight
Tungro
Tungro
Bacterialblight
Tungro
Blast
Brownspot
Tungro
Blast
Blast
Blast
Bacterialblight
Tungro
Blast
Bacterialblight
Bacterialblight

```
Training will proceed until epoch 10 then you will be asked to
enter H to halt training or enter an integer for how many more epochs to run then be asked again
learning rate will be automatically adjusted during training
 Epoch   Train    Train    Valid    Valid     V_Loss %    Learning    Next LR   Duration in
         Loss   Accuracy   Loss    Accuracy  Improvement    Rate        Rate      Seconds
   1    6.2302    92.63   5.8153    68.12        0.00      0.001000    0.001000    47.75
   2    4.0382    99.87   3.8419    75.42       33.93      0.001000    0.001000    15.58
   3    2.7367    99.82   2.7654    78.33       28.02      0.001000    0.001000    15.32
   4    1.8964    99.97   2.0067    82.08       27.44      0.001000    0.001000    15.29
   5    1.3412    99.97   1.4581    86.87       27.34      0.001000    0.001000    15.38
   6    0.9612    99.97   1.1328    86.67       22.31      0.001000    0.001000    15.23
   7    0.6997   100.00   0.8419    90.00       25.68      0.001000    0.001000    15.23
   8    0.5180   100.00   0.5998    94.17       28.76      0.001000    0.001000    15.32
   9    0.3899   100.00   0.4347    96.88       27.52      0.001000    0.001000    15.27
  10    0.3017   100.00   0.3442    97.29       20.81      0.001000    0.001000    15.22

 Enter H to end training or  an integer for the number of additional epochs to run then ask again
you entered 5 Training will continue to epoch 15
 Epoch   Train    Train    Valid    Valid     V_Loss %    Learning    Next LR   Duration in
         Loss   Accuracy   Loss    Accuracy  Improvement    Rate        Rate      Seconds
  11    0.2425   100.00   0.2707    98.54       21.37      0.001000    0.001000    15.25
  12    0.2010   100.00   0.2043    99.58       24.50      0.001000    0.001000    15.33
  13    0.1743   100.00   0.1629   100.00       20.26      0.001000    0.001000    15.45
  14    0.1561   100.00   0.1586    99.58        2.64      0.001000    0.001000    15.17
  15    0.1415   100.00   0.1253   100.00       21.02      0.001000    0.001000    15.38

 Enter H to end training or  an integer for the number of additional epochs to run then ask again
you entered h,  Training halted on epoch 15 due to user input

loading model with weights from epoch 15
training elapsed time was 0.0 hours,  4.0 minutes, 34.04 seconds)
```



To save the training data to a csv file enter the name for the csv file or press enter to not save the data

```
6/6 [==============================] - 3s 152ms/step
There were 0 errors in 480 tests for an accuracy of 100.00 and an F1 score of 10
0.00
```




```
Classification Report:
----------------------
                  precision    recall  f1-score   support

Bacterialblight     1.0000    1.0000    1.0000       120
          Blast     1.0000    1.0000    1.0000       120
      Brownspot     1.0000    1.0000    1.0000       120
         Tungro     1.0000    1.0000    1.0000       120

       accuracy                         1.0000       480
      macro avg     1.0000    1.0000    1.0000       480
   weighted avg     1.0000    1.0000    1.0000       480
```

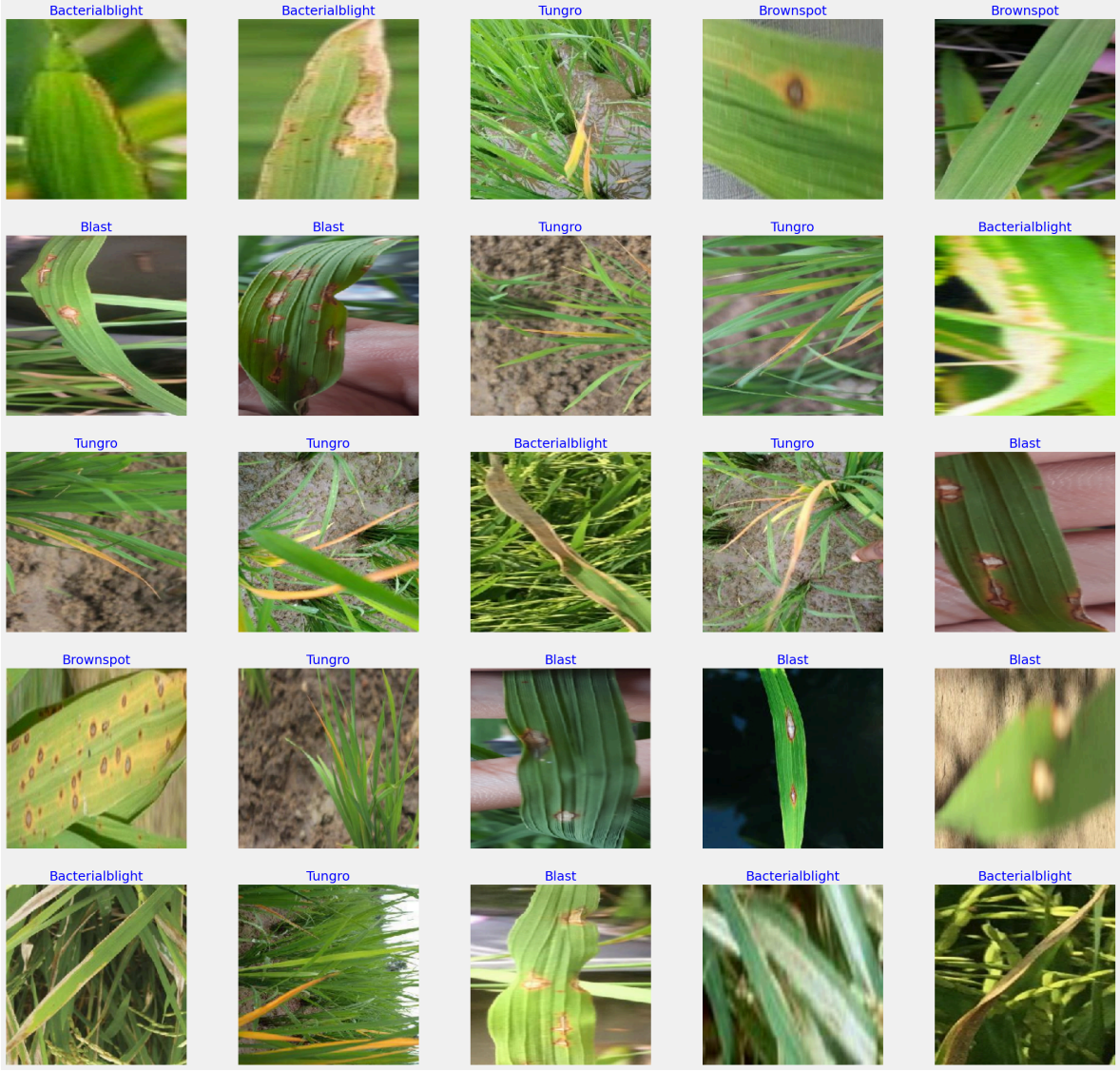
your trained model will be saved to directory /kaggle/working/ enter a subject for the saved model

model was saved as /kaggle/working/MobileNetV3 large-4-(224 X 224)- 100.00.h5
model save nomenclature is directory/subject-number of classes- (img_height, img_width)- F1score.h5
enter R to rerun the classifier or press enter to quit
Enter N to specify new data directories or hit Enter to use existing data directory paths
the train directory contains 5932 files, class Brownspot has the most images of 1600 files
class Tungro contains the least number of images of 1308 files
NOTE if the value of most images is the same as the value of least images the dataset is balanced
When dealing with very large data sets to save time you may not want to read in all the image files
to limit the maximum number of image files in any class you can enter an integer limiter value
input a limiter integer value to limit max number of images in a class, for no limit hit enter
images will be limited to a maximum of 1200 images in each class
enter the minimum number of training images a class must have to be included in the dataset or press enter to use the default value of 10
processing images in train directory for class Tungro
number of classes in processed dataset= 4
the maximum files in any class in train_df is 960 the minimum files in any class in train_df is 960
train_df length: 3840 test_df length: 480 valid_df length: 480
All image files were properly processed and used in the dataframes

the average height of training images is 313 and average width is 314
enter the image height to be used to train the model
Enter the image width to be used to train the model
model will be trained with image shape of (224, 224)
enter A to auto balance the train set or enter or hit enter to leave train set unchanged
training data set will be used as is
enter the desired batch size or press enter to use the default batch size of 20
select a model number from the list below

Model No	Model Type	Parameters
1	MobileNetV3-small	1.0 M
2	MobileNetV3-large	3.3 M
3	EfficientNetV2B0	6.3 M
4	EfficientNetV2B1	7.3 M
5	EfficientNetV2B2	9.1 M
6	InceptionResNetV2	54.7 M

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-application
s/efficientnet_v2/efficientnetv2-b2_notop.h5
35839040/35839040 [==============================] - 0s 0us/step
Created EfficientNetV2 B2 model with initial learning rate set to 0.001
enter the number of epochs to run initially. After these epochs complete you can
enter H to halt training
or enter an integer for how many more epochs to run then be asked again
Found 3840 validated image filenames belonging to 4 classes.          for train
generator
Found 480 validated image filenames belonging to 4 classes.           for valid
generator
Found 480 validated image filenames belonging to 4 classes.           for test g
enerator
Below are some example training images
```

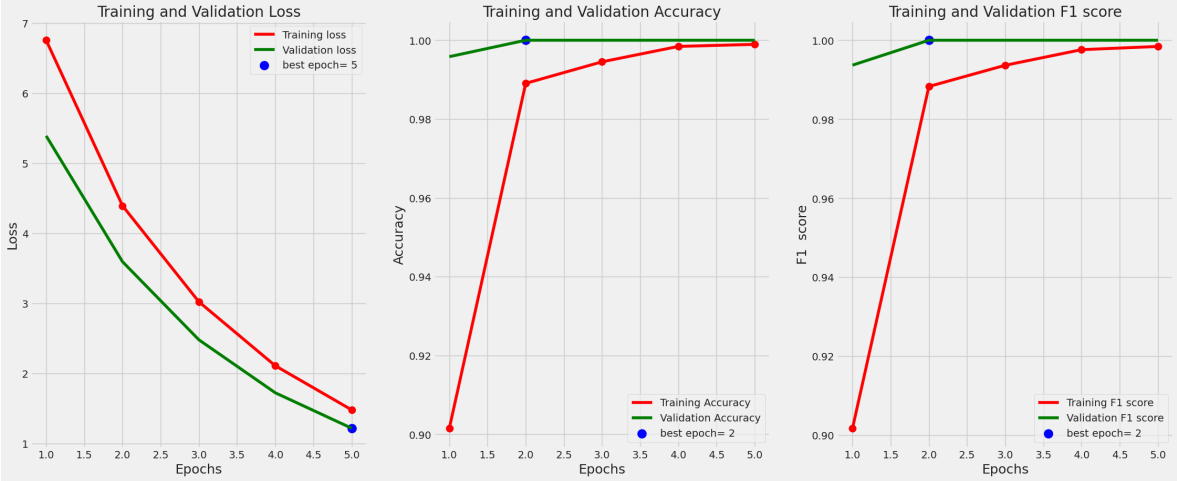



```
Training will proceed until epoch 5 then you will be asked to
enter H to halt training or enter an integer for how many more epochs to run then
be asked again
learning rate will be automatically adjusted during training
2024-01-01 12:24:08.921426: E tensorflow/core/grappler/optimizers/meta_optimizer.
cc:954] layout failed: INVALID_ARGUMENT: Size of values 0 does not match size of
permutation 4 @ fanin shape inmodel_3/block1b_drop/dropout/SelectV2-2-TransposeNH
WCToNCHW-LayoutOptimizer
```

Epoch	Train Loss	Train Accuracy	Valid Loss	Valid Accuracy	V_Loss % Improvement	Learning Rate	Next LR	Duration in Seconds
1	6.7565	90.16	5.3913	99.58	0.00	0.001000	0.001000	89.09
2	4.3933	98.91	3.5943	100.00	33.33	0.001000	0.001000	32.99
3	3.0224	99.45	2.4798	100.00	31.01	0.001000	0.001000	33.04
4	2.1098	99.84	1.7236	100.00	30.49	0.001000	0.001000	33.02
5	1.4799	99.90	1.2177	100.00	29.35	0.001000	0.001000	33.22

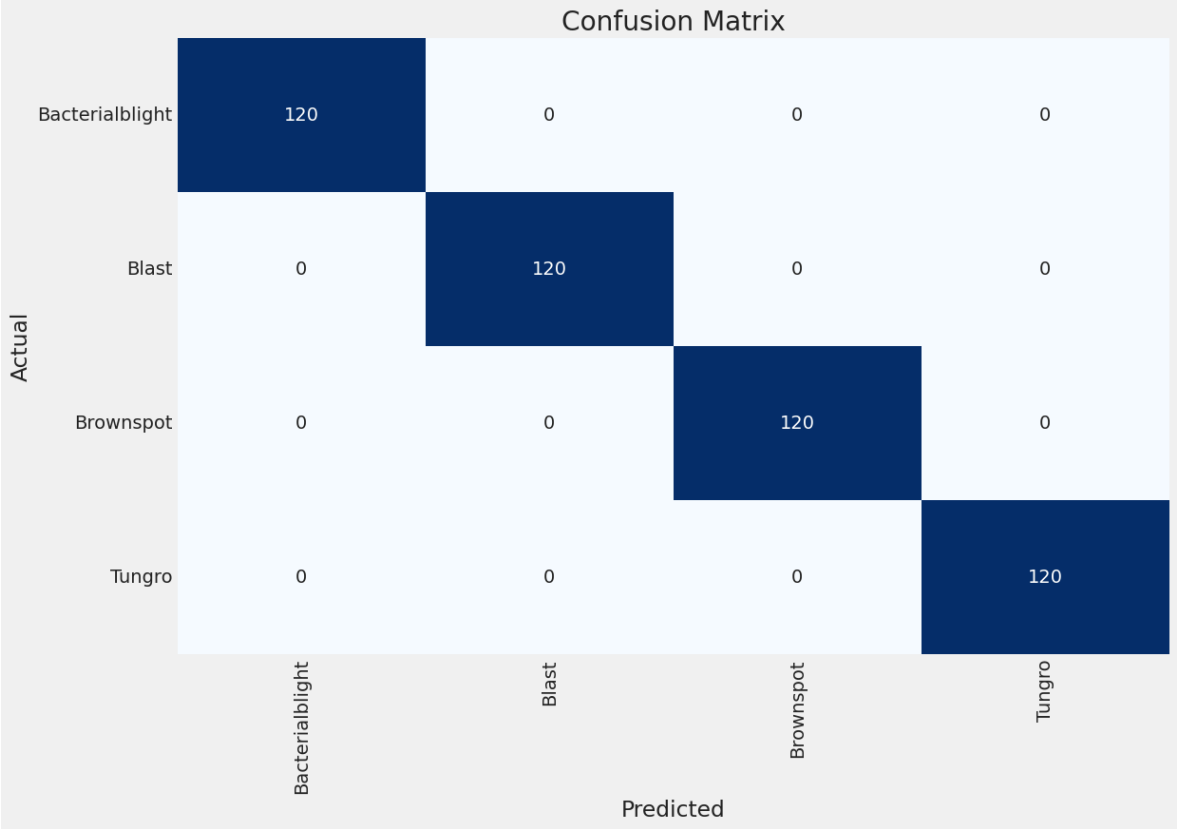
Enter H to end training or an integer for the number of additional epochs to run then ask again
you entered h, Training halted on epoch 5 due to user input

loading model with weights from epoch 5
training elapsed time was 0.0 hours, 5.0 minutes, 31.09 seconds)



To save the training data to a csv file enter the name for the csv file or press enter to not save the data

```
6/6 [==============================] - 4s 181ms/step
There were 0 errors in 480 tests for an accuracy of 100.00 and an F1 score of 100.00
```




```
Classification Report:
----------------------
                  precision    recall  f1-score   support

Bacterialblight     1.0000    1.0000    1.0000       120
          Blast     1.0000    1.0000    1.0000       120
      Brownspot     1.0000    1.0000    1.0000       120
         Tungro     1.0000    1.0000    1.0000       120

       accuracy                         1.0000       480
      macro avg     1.0000    1.0000    1.0000       480
   weighted avg     1.0000    1.0000    1.0000       480
```

```
your trained model will be saved to directory /kaggle/working/ enter a subject fo
r the saved model
model was saved as /kaggle/working/EfficientNetB2-4-(224 X 224)- 100.00.h5
model save nomenclature is directory/subject-number of classes- (img_height, img_
width)- F1score.h5
enter R to rerun the classifier or press enter to quit
process commpleted
```

Model Evaluation

4 models were run with the same set of input image parameters. A limiter was set so that each class was limited to 1200 images. An image size of 224 X 224 was used. Since all classes have more than 1200 images the limited dataset is balanced and no image augmentation was required. Batch size was set to 40. The first model run was an EfficientNetB0 model. This model converged very quicklyachieving a validation accuracy of 100% in

just 2 epochs. After 10 epochs the model achieved an F1 score of 100% with a total training time of 4 min and 19 seconds. The 2nd model run was a MobileNetV3 small model. This model converged slowly. After 10 epochs it achieved a validation accuracy of 91.7% After 15 epochs it achieved a validation accuracy of 100% The model was run for a total of 20 epochs and achieved an F1 score of 100% with a total run time of 4 minutes and 19 seconds. The 3rd model run was an MobileNetV3 large model. This model was run for 15 epochs and achieved an F1 score of 100% with a total run time of 4 minutes and 34 seconds. The final model run was an EfficientNetB2 model. This model converged very quicly. It was run for 5 epochs abd achieved an F1 score of 100% with a run time of 5 minutes and 31 seconds.