pre-final

May 4, 2024

[2]:

```
# Import libraries
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from keras.applications import MobileNetV2
from keras.preprocessing.image import ImageDataGenerator
import numpy as np
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import cv2
import pandas as pd

import cv2
import numpy as np
import pandas as pd
import seaborn as sns
sns.set_style('darkgrid')
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam, Adamax
from tensorflow.keras.metrics import categorical_crossentropy
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,␣
 ↪Activation, Dropout, BatchNormalization
from tensorflow.keras import regularizers

# Ignore Warnings
import warnings
warnings.filterwarnings("ignore")

import os
import time
```

```python
import shutil
import pathlib
import itertools
```

```
/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A
NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy
(detected version 1.23.5
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

[3]:

```python
# Generate data paths with labels
def define_paths(data_dir):
    filepaths = []
    labels = []

    folds = os.listdir(data_dir)
    for fold in folds:
        foldpath = os.path.join(data_dir, fold)
        # check the folders from main directory. If there are another files,␣
↪ignore them
        if pathlib.Path(foldpath).suffix != '':
            continue

        filelist = os.listdir(foldpath)
        for file in filelist:
            fpath = os.path.join(foldpath, file)

            # check if there are another folders
            if pathlib.Path(foldpath).suffix == '':
                # check unneeded masks
                if pathlib.Path(fpath).parts[-1] == 'masks' or pathlib.
↪Path(fpath).parts[-1] == 'Masks' or pathlib.Path(fpath).parts[-1] == 'MASKS':
                    continue

                else:
                    o_file = os.listdir(fpath)
                    for f in o_file:
                        ipath = os.path.join(fpath, f)
                        filepaths.append(ipath)
                        labels.append(fold)

            else:
                filepaths.append(fpath)
                labels.append(fold)

    return filepaths, labels
```

```python
# Concatenate data paths with labels into one dataframe ( to later be fitted into the model )
def define_df(files, classes):
    Fseries = pd.Series(files, name= 'filepaths')
    Lseries = pd.Series(classes, name='labels')
    return pd.concat([Fseries, Lseries], axis= 1)

# Split dataframe to train, valid, and test
def split_data(data_dir):
    # train dataframe
    files, classes = define_paths(data_dir)
    df = define_df(files, classes)
    strat = df['labels']
    train_df, dummy_df = train_test_split(df,  train_size= 0.8, shuffle= True, random_state= 123, stratify= strat)

    # valid and test dataframe
    strat = dummy_df['labels']
    valid_df, test_df = train_test_split(dummy_df,  train_size= 0.5, shuffle= True, random_state= 123, stratify= strat)

    return train_df, valid_df, test_df
```

[4]:

```python
def create_gens (train_df, valid_df, test_df, batch_size):
    '''
    This function takes train, validation, and test dataframe and fit them into image data generator, because model takes data from image data generator.
    Image data generator converts images into tensors. '''


    # define model parameters
    img_size = (224, 224)
    channels = 3 # either BGR or Grayscale
    color = 'rgb'
    img_shape = (img_size[0], img_size[1], channels)

    # Recommended : use custom function for test data batch size, else we can use normal batch size.
    ts_length = len(test_df)
    test_batch_size = max(sorted([ts_length // n for n in range(1, ts_length + 1) if ts_length%n == 0 and ts_length/n <= 80]))
    test_steps = ts_length // test_batch_size

    # This function which will be used in image data generator for data augmentation, it just take the image and return it again.
    def scalar(img):
        return img

    tr_gen = ImageDataGenerator(preprocessing_function= scalar,␣
↪horizontal_flip= True)
    ts_gen = ImageDataGenerator(preprocessing_function= scalar)

    train_gen = tr_gen.flow_from_dataframe( train_df, x_col= 'filepaths',␣
↪y_col= 'labels', target_size= img_size, class_mode= 'categorical',
                                        color_mode= color, shuffle= True,␣
↪batch_size= batch_size)

    valid_gen = ts_gen.flow_from_dataframe( valid_df, x_col= 'filepaths',␣
↪y_col= 'labels', target_size= img_size, class_mode= 'categorical',
                                        color_mode= color, shuffle= True,␣
↪batch_size= batch_size)

    # Note: we will use custom test_batch_size, and make shuffle= false
    test_gen = ts_gen.flow_from_dataframe( test_df, x_col= 'filepaths', y_col=␣
↪'labels', target_size= img_size, class_mode= 'categorical',
                                        color_mode= color, shuffle= False,␣
↪batch_size= test_batch_size)

    return train_gen, valid_gen, test_gen
```

[5]:

```
def plot_training(hist):
    '''
    This function take training model and plot history of accuracy and losses␣
↪with the best epoch in both of them.
    '''

    # Define needed variables
    tr_acc = hist.history['accuracy']
    tr_loss = hist.history['loss']
    val_acc = hist.history['val_accuracy']
    val_loss = hist.history['val_loss']
    index_loss = np.argmin(val_loss)
    val_lowest = val_loss[index_loss]
    index_acc = np.argmax(val_acc)
    acc_highest = val_acc[index_acc]
    Epochs = [i+1 for i in range(len(tr_acc))]
    loss_label = f'best epoch= {str(index_loss + 1)}'
    acc_label = f'best epoch= {str(index_acc + 1)}'

    # Plot training history
    plt.figure(figsize= (20, 8))
    plt.style.use('fivethirtyeight')
    plt.subplot(1, 2, 1)
    plt.plot(Epochs, tr_loss, 'r', label= 'Training loss')
    plt.plot(Epochs, val_loss, 'g', label= 'Validation loss')
    plt.scatter(index_loss + 1, val_lowest, s= 150, c= 'blue', label=␣
 ↪loss_label)
    plt.title('Training and Validation Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(Epochs, tr_acc, 'r', label= 'Training Accuracy')
    plt.plot(Epochs, val_acc, 'g', label= 'Validation Accuracy')
    plt.scatter(index_acc + 1 , acc_highest, s= 150, c= 'blue', label=␣
 ↪acc_label)
    plt.title('Training and Validation Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()

    plt.tight_layout
    plt.show()
```

[35]:

```
def plot_confusion_matrix(cm, classes, normalize= False, title= 'Confusion␣
 ↪Matrix', cmap= plt.cm.Greens):
        '''
        This function plot confusion matrix method from sklearn package.
        '''

        plt.figure(figsize= (10, 10))
        plt.imshow(cm, interpolation= 'nearest', cmap= cmap)
        plt.title(title)
        plt.colorbar()

        tick_marks = np.arange(len(classes))
        plt.xticks(tick_marks, classes, rotation= 45)
        plt.yticks(tick_marks, classes)

        if normalize:
                cm = cm.astype('float') / cm.sum(axis= 1)[:, np.newaxis]
                print('Normalized Confusion Matrix')

        else:
                print('Confusion Matrix, Without Normalization')

        print(cm)
        thresh = cm.max() / 2.
        for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
                plt.text(j, i, cm[i, j], horizontalalignment= 'center', color=␣
 ↪'white' if cm[i, j] > thresh else 'black')

        plt.tight_layout()
        plt.ylabel('True Label')
        plt.xlabel('Predicted Label')
```

[7]:

```
def show_images(gen):
    '''
    This function take the data generator and show sample of the images
    '''

    # return classes , images to be displayed
    g_dict = gen.class_indices        # defines dictionary {'class': index}
    classes = list(g_dict.keys())     # defines list of dictionary's kays␣
 ↪(classes), classes names : string
    images, labels = next(gen)        # get a batch size samples from the␣
 ↪generator

    # calculate number of displayed samples
    length = len(labels)        # length of batch size
    sample = min(length, 25)    # check if sample less than 25 images

    plt.figure(figsize= (20, 20))

    for i in range(sample):
        plt.subplot(5, 5, i + 1)
        image = images[i] / 255       # scales data to range (0 - 255)
        plt.imshow(image)
        index = np.argmax(labels[i])  # get image index
        class_name = classes[index]   # get class of image
        plt.title(class_name, color= 'blue', fontsize= 12)
        plt.axis('off')
    plt.show()
```

[8]:

```
class MyCallback(keras.callbacks.Callback):
    def __init__(self, model, patience, stop_patience, threshold, factor,␣
 ↪batches, epochs, ask_epoch):
        super(MyCallback, self).__init__()
        self.model = model

        self.patience = patience # specifies how many epochs without␣
 ↪improvement before learning rate is adjusted
```

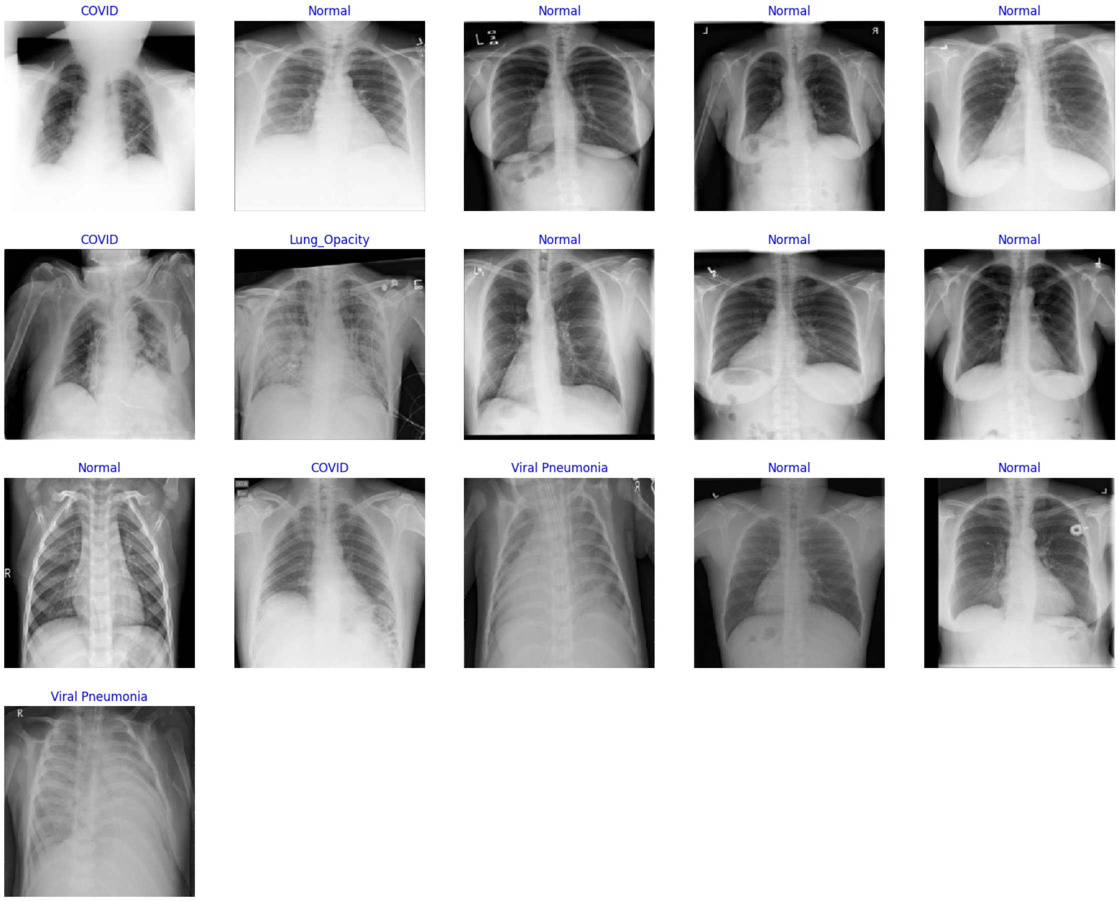
```
        self.stop_patience = stop_patience # specifies how many times to adjust lr without improvement to stop training
        self.threshold = threshold # specifies training accuracy threshold when lr will be adjusted based on validation loss
        self.factor = factor # factor by which to reduce the learning rate
        self.batches = batches # number of training batch to run per epoch
        self.epochs = epochs
        self.ask_epoch = ask_epoch
        self.ask_epoch_initial = ask_epoch # save this value to restore if restarting training

        # callback variables
        self.count = 0 # how many times lr has been reduced without improvement
        self.stop_count = 0
        self.best_epoch = 1    # epoch with the lowest loss
        self.initial_lr = float(tf.keras.backend.get_value(model.optimizer.lr)) # get the initial learning rate and save it
        self.highest_tracc = 0.0 # set highest training accuracy to 0 initially
        self.lowest_vloss = np.inf # set lowest validation loss to infinity initially
        self.best_weights = self.model.get_weights() # set best weights to model's initial weights
        self.initial_weights = self.model.get_weights()   # save initial weights if they have to get restored

    # Define a function that will run when train begins
    def on_train_begin(self, logs= None):
        msg = 'Do you want model asks you to halt the training [y/n] ?'
        print(msg)
        ans = "n" #input('')
        if ans in ['Y', 'y']:
            self.ask_permission = 1
        elif ans in ['N', 'n']:
            self.ask_permission = 0

        msg = '{0:^8s}{1:^10s}{2:^9s}{3:^9s}{4:^9s}{5:^9s}{6:^9s}{7:^10s}{8:10s}{9:^8s}'.format('Epoch', 'Loss', 'Accuracy', 'V_loss', 'V_acc', 'LR', 'Next LR', 'Monitor','% Improv', 'Duration')
        print(msg)
        self.start_time = time.time()


    def on_train_end(self, logs= None):
        stop_time = time.time()
        tr_duration = stop_time - self.start_time
        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}␣
↪minutes, {seconds:4.2f} seconds)'
        print(msg)

        # set the weights of the model to the best weights
        self.model.set_weights(self.best_weights)


    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}   ␣
↪-   loss: {4:8.5f}'.format(' ', str(batch), str(self.batches), acc, loss)
        print(msg, '\r', end= '')


    def on_epoch_begin(self, epoch, logs= None):
        self.ep_start = time.time()


    # Define method runs on the end of each epoch
    def on_epoch_end(self, epoch, logs= None):
        ep_end = time.time()
        duration = ep_end - self.ep_start

        lr = float(tf.keras.backend.get_value(self.model.optimizer.lr)) # get␣
↪the current learning rate
        current_lr = lr
        acc = logs.get('accuracy')  # get training accuracy
        v_acc = logs.get('val_accuracy')  # get validation accuracy
        loss = logs.get('loss')  # get training loss for this epoch
        v_loss = logs.get('val_loss')  # get the validation loss for this epoch

        if acc < self.threshold: # if training accuracy is below threshold␣
↪adjust lr based on training accuracy
            monitor = 'accuracy'
            if epoch == 0:
                pimprov = 0.0
            else:
                pimprov = (acc - self.highest_tracc ) * 100 / self.
↪highest_tracc # define improvement of model progres
```

```
            if acc > self.highest_tracc: # training accuracy improved in the epoch
                self.highest_tracc = acc # set new highest training accuracy
                self.best_weights = self.model.get_weights() # training accuracy improved so save the weights
                self.count = 0 # set count to 0 since training accuracy improved
                self.stop_count = 0 # set stop counter to 0
                if v_loss < self.lowest_vloss:
                    self.lowest_vloss = v_loss
                self.best_epoch = epoch + 1  # set the value of best epoch for this epoch

            else:
                # training accuracy did not improve check if this has happened for patience number of epochs
                # if so adjust learning rate
                if self.count >= self.patience - 1: # lr should be adjusted
                    lr = lr * self.factor # adjust the learning by factor
                    tf.keras.backend.set_value(self.model.optimizer.lr, lr) # set the learning rate in the optimizer
                    self.count = 0 # reset the count to 0
                    self.stop_count = self.stop_count + 1 # count the number of consecutive lr adjustments
                    self.count = 0 # reset counter
                    if v_loss < self.lowest_vloss:
                        self.lowest_vloss = v_loss
                else:
                    self.count = self.count + 1 # increment patience counter

        else: # training accuracy is above threshold so adjust learning rate based on validation loss
            monitor = 'val_loss'
            if epoch == 0:
                pimprov = 0.0

            else:
                pimprov = (self.lowest_vloss - v_loss ) * 100 / self.lowest_vloss

            if v_loss < self.lowest_vloss: # check if the validation loss improved
                self.lowest_vloss = v_loss # replace lowest validation loss with new validation loss
                self.best_weights = self.model.get_weights() # validation loss improved so save the weights
                self.count = 0 # reset count since validation loss improved
                self.stop_count = 0
                self.best_epoch = epoch + 1 # set the value of the best epoch␣
↪to this epoch

            else: # validation loss did not improve
                if self.count >= self.patience - 1: # need to adjust lr
                    lr = lr * self.factor # adjust the learning rate
                    self.stop_count = self.stop_count + 1 # increment stop␣
↪counter because lr was adjusted
                    self.count = 0 # reset counter
                    tf.keras.backend.set_value(self.model.optimizer.lr, lr) #␣
↪set the learning rate in the optimizer

                else:
                    self.count = self.count + 1 # increment the patience counter

                if acc > self.highest_tracc:
                    self.highest_tracc = acc

        msg = f'{str(epoch + 1):^3s}/{str(self.epochs):4s} {loss:^9.3f}{acc *␣
↪100:^9.3f}{v_loss:^9.5f}{v_acc * 100:^9.3f}{current_lr:^9.5f}{lr:^9.
↪5f}{monitor:^11s}{pimprov:^10.2f}{duration:^8.2f}'
        print(msg)

        if self.stop_count > self.stop_patience - 1: # check if learning rate␣
↪has been adjusted stop_count times with no improvement
            msg = f' training has been halted at epoch {epoch + 1} after {self.
↪stop_patience} adjustments of learning rate with no improvement'
            print(msg)
            self.model.stop_training = True # stop training

        else:
            if self.ask_epoch != None and self.ask_permission != 0:
                if epoch + 1 >= self.ask_epoch:
                    msg = 'enter H to halt training or an integer for number of␣
↪epochs to run then ask again'
                    print(msg)

                    ans = input('')
                    if ans == 'H' or ans == 'h':
                        msg = f'training has been halted at epoch {epoch + 1}␣
↪due to user input'
                        print(msg)
                        self.model.stop_training = True # stop training
                    else:
                        try:
                            ans = int(ans)
                            self.ask_epoch += ans
                            msg = f' training will continue until epoch␣
 ↪{str(self.ask_epoch)}'
                            print(msg)
                            msg = '{0:^8s}{1:^10s}{2:^9s}{3:^9s}{4:^9s}{5:
 ↪^9s}{6:^9s}{7:^10s}{8:10s}{9:^8s}'.format('Epoch', 'Loss', 'Accuracy',␣
 ↪'V_loss', 'V_acc', 'LR', 'Next LR', 'Monitor', '% Improv', 'Duration')
                            print(msg)

                        except Exception:
                            print('Invalid')
```

[9]:
```
data_dir = '/kaggle/input/covid19-radiography-database/
 ↪COVID-19_Radiography_Dataset'

try:
    # Get splitted data
    train_df, valid_df, test_df = split_data(data_dir)

    # Get Generators
    batch_size = 16
    train_gen, valid_gen, test_gen = create_gens(train_df, valid_df, test_df,␣
 ↪batch_size)

except:
    print('Invalid Input')
```

```
Found 16932 validated image filenames belonging to 4 classes.
Found 2116 validated image filenames belonging to 4 classes.
Found 2117 validated image filenames belonging to 4 classes.
```

[10]:
```
show_images(train_gen)
```



[11]:

```
#3 & 4. Load Pre-trained MobileNetV2

img_size = (224, 224)
channels = 3
img_shape = (img_size[0], img_size[1], channels)

def getBaseModel():
    base_model = MobileNetV2(
    weights="imagenet", include_top=False, input_shape= img_shape
    )
    return base_model
base_model = getBaseModel()
base_model.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applicatio
ns/mobilenet_v2/mobilenet_v2_weights_tf_dim_ordering_tf_kernels_1.0_224_no_top.h
5
9406464/9406464 [==============================] - 0s 0us/step
Model: "mobilenetv2_1.00_224"
```

```
--------------------------------------------------------------------------------
------------------
 Layer (type)                   Output Shape         Param #     Connected to
================================================================================
==================
 input_1 (InputLayer)           [(None, 224, 224, 3  0           []
                                )]

 Conv1 (Conv2D)                 (None, 112, 112, 32  864
['input_1[0][0]']
                                )

 bn_Conv1 (BatchNormalization)  (None, 112, 112, 32  128         ['Conv1[0][0]']
                                )

 Conv1_relu (ReLU)              (None, 112, 112, 32  0
['bn_Conv1[0][0]']
                                )

 expanded_conv_depthwise (Depth  (None, 112, 112, 32  288
['Conv1_relu[0][0]']
 wiseConv2D)                    )

 expanded_conv_depthwise_BN (Ba  (None, 112, 112, 32  128
['expanded_conv_depthwise[0][0]']
 tchNormalization)              )

 expanded_conv_depthwise_relu (  (None, 112, 112, 32  0
['expanded_conv_depthwise_BN[0][0
 ReLU)                          )                                ]']

 expanded_conv_project (Conv2D)  (None, 112, 112, 16  512
['expanded_conv_depthwise_relu[0]
                                )                                [0]']

 expanded_conv_project_BN (Batc  (None, 112, 112, 16  64
['expanded_conv_project[0][0]']
 hNormalization)                )

 block_1_expand (Conv2D)        (None, 112, 112, 96  1536
['expanded_conv_project_BN[0][0]'
                                )                                ]

 block_1_expand_BN (BatchNormal  (None, 112, 112, 96  384
['block_1_expand[0][0]']
 ization)                       )

 block_1_expand_relu (ReLU)     (None, 112, 112, 96  0
['block_1_expand_BN[0][0]']
                                )

 block_1_pad (ZeroPadding2D)    (None, 113, 113, 96  0
['block_1_expand_relu[0][0]']
                                )

 block_1_depthwise (DepthwiseCo  (None, 56, 56, 96)  864
['block_1_pad[0][0]']
 nv2D)

 block_1_depthwise_BN (BatchNor  (None, 56, 56, 96)  384
['block_1_depthwise[0][0]']
 malization)

 block_1_depthwise_relu (ReLU)  (None, 56, 56, 96)   0
['block_1_depthwise_BN[0][0]']

 block_1_project (Conv2D)       (None, 56, 56, 24)   2304
['block_1_depthwise_relu[0][0]']

 block_1_project_BN (BatchNorma  (None, 56, 56, 24)  96
['block_1_project[0][0]']
 lization)

 block_2_expand (Conv2D)        (None, 56, 56, 144)  3456
['block_1_project_BN[0][0]']

 block_2_expand_BN (BatchNormal  (None, 56, 56, 144)  576
['block_2_expand[0][0]']
 ization)

 block_2_expand_relu (ReLU)     (None, 56, 56, 144)  0
['block_2_expand_BN[0][0]']

 block_2_depthwise (DepthwiseCo  (None, 56, 56, 144)  1296
['block_2_expand_relu[0][0]']
 nv2D)

 block_2_depthwise_BN (BatchNor  (None, 56, 56, 144)  576
['block_2_depthwise[0][0]']
 malization)

 block_2_depthwise_relu (ReLU)  (None, 56, 56, 144)  0
['block_2_depthwise_BN[0][0]']

 block_2_project (Conv2D)       (None, 56, 56, 24)   3456
['block_2_depthwise_relu[0][0]']
```

```
 block_2_project_BN (BatchNorma  (None, 56, 56, 24)  96
['block_2_project[0][0]']
 lization)

 block_2_add (Add)              (None, 56, 56, 24)   0
['block_1_project_BN[0][0]',
'block_2_project_BN[0][0]']

 block_3_expand (Conv2D)        (None, 56, 56, 144)  3456
['block_2_add[0][0]']

 block_3_expand_BN (BatchNormal  (None, 56, 56, 144)  576
['block_3_expand[0][0]']
 ization)

 block_3_expand_relu (ReLU)     (None, 56, 56, 144)  0
['block_3_expand_BN[0][0]']

 block_3_pad (ZeroPadding2D)    (None, 57, 57, 144)  0
['block_3_expand_relu[0][0]']

 block_3_depthwise (DepthwiseCo  (None, 28, 28, 144)  1296
['block_3_pad[0][0]']
 nv2D)

 block_3_depthwise_BN (BatchNor  (None, 28, 28, 144)  576
['block_3_depthwise[0][0]']
 malization)

 block_3_depthwise_relu (ReLU)  (None, 28, 28, 144)  0
['block_3_depthwise_BN[0][0]']

 block_3_project (Conv2D)       (None, 28, 28, 32)   4608
['block_3_depthwise_relu[0][0]']

 block_3_project_BN (BatchNorma  (None, 28, 28, 32)  128
['block_3_project[0][0]']
 lization)

 block_4_expand (Conv2D)        (None, 28, 28, 192)  6144
['block_3_project_BN[0][0]']

 block_4_expand_BN (BatchNormal  (None, 28, 28, 192)  768
['block_4_expand[0][0]']
 ization)

 block_4_expand_relu (ReLU)     (None, 28, 28, 192)  0
```

```
['block_4_expand_BN[0][0]']

 block_4_depthwise (DepthwiseCo  (None, 28, 28, 192)  1728
['block_4_expand_relu[0][0]']
 nv2D)

 block_4_depthwise_BN (BatchNor  (None, 28, 28, 192)  768
['block_4_depthwise[0][0]']
 malization)

 block_4_depthwise_relu (ReLU)  (None, 28, 28, 192)  0
['block_4_depthwise_BN[0][0]']

 block_4_project (Conv2D)       (None, 28, 28, 32)   6144
['block_4_depthwise_relu[0][0]']

 block_4_project_BN (BatchNorma  (None, 28, 28, 32)  128
['block_4_project[0][0]']
 lization)

 block_4_add (Add)              (None, 28, 28, 32)   0
['block_3_project_BN[0][0]',
'block_4_project_BN[0][0]']

 block_5_expand (Conv2D)        (None, 28, 28, 192)  6144
['block_4_add[0][0]']

 block_5_expand_BN (BatchNormal  (None, 28, 28, 192)  768
['block_5_expand[0][0]']
 ization)

 block_5_expand_relu (ReLU)     (None, 28, 28, 192)  0
['block_5_expand_BN[0][0]']

 block_5_depthwise (DepthwiseCo  (None, 28, 28, 192)  1728
['block_5_expand_relu[0][0]']
 nv2D)

 block_5_depthwise_BN (BatchNor  (None, 28, 28, 192)  768
['block_5_depthwise[0][0]']
 malization)

 block_5_depthwise_relu (ReLU)  (None, 28, 28, 192)  0
['block_5_depthwise_BN[0][0]']

 block_5_project (Conv2D)       (None, 28, 28, 32)   6144
['block_5_depthwise_relu[0][0]']
```

```
 block_5_project_BN (BatchNorma  (None, 28, 28, 32)  128
['block_5_project[0][0]']
 lization)

 block_5_add (Add)              (None, 28, 28, 32)   0
['block_4_add[0][0]',
'block_5_project_BN[0][0]']

 block_6_expand (Conv2D)        (None, 28, 28, 192)  6144
['block_5_add[0][0]']

 block_6_expand_BN (BatchNormal  (None, 28, 28, 192)  768
['block_6_expand[0][0]']
 ization)

 block_6_expand_relu (ReLU)     (None, 28, 28, 192)  0
['block_6_expand_BN[0][0]']

 block_6_pad (ZeroPadding2D)    (None, 29, 29, 192)  0
['block_6_expand_relu[0][0]']

 block_6_depthwise (DepthwiseCo  (None, 14, 14, 192)  1728
['block_6_pad[0][0]']
 nv2D)

 block_6_depthwise_BN (BatchNor  (None, 14, 14, 192)  768
['block_6_depthwise[0][0]']
 malization)

 block_6_depthwise_relu (ReLU)  (None, 14, 14, 192)  0
['block_6_depthwise_BN[0][0]']

 block_6_project (Conv2D)       (None, 14, 14, 64)   12288
['block_6_depthwise_relu[0][0]']

 block_6_project_BN (BatchNorma  (None, 14, 14, 64)  256
['block_6_project[0][0]']
 lization)

 block_7_expand (Conv2D)        (None, 14, 14, 384)  24576
['block_6_project_BN[0][0]']

 block_7_expand_BN (BatchNormal  (None, 14, 14, 384)  1536
['block_7_expand[0][0]']
 ization)

 block_7_expand_relu (ReLU)     (None, 14, 14, 384)  0
['block_7_expand_BN[0][0]']
```

```
 block_7_depthwise (DepthwiseCo  (None, 14, 14, 384)  3456
['block_7_expand_relu[0][0]']
 nv2D)

 block_7_depthwise_BN (BatchNor  (None, 14, 14, 384)  1536
['block_7_depthwise[0][0]']
 malization)

 block_7_depthwise_relu (ReLU)  (None, 14, 14, 384)  0
['block_7_depthwise_BN[0][0]']

 block_7_project (Conv2D)       (None, 14, 14, 64)   24576
['block_7_depthwise_relu[0][0]']

 block_7_project_BN (BatchNorma  (None, 14, 14, 64)  256
['block_7_project[0][0]']
 lization)

 block_7_add (Add)              (None, 14, 14, 64)   0
['block_6_project_BN[0][0]',
'block_7_project_BN[0][0]']

 block_8_expand (Conv2D)        (None, 14, 14, 384)  24576
['block_7_add[0][0]']

 block_8_expand_BN (BatchNormal  (None, 14, 14, 384)  1536
['block_8_expand[0][0]']
 ization)

 block_8_expand_relu (ReLU)     (None, 14, 14, 384)  0
['block_8_expand_BN[0][0]']

 block_8_depthwise (DepthwiseCo  (None, 14, 14, 384)  3456
['block_8_expand_relu[0][0]']
 nv2D)

 block_8_depthwise_BN (BatchNor  (None, 14, 14, 384)  1536
['block_8_depthwise[0][0]']
 malization)

 block_8_depthwise_relu (ReLU)  (None, 14, 14, 384)  0
['block_8_depthwise_BN[0][0]']

 block_8_project (Conv2D)       (None, 14, 14, 64)   24576
['block_8_depthwise_relu[0][0]']

 block_8_project_BN (BatchNorma  (None, 14, 14, 64)  256
['block_8_project[0][0]']
 lization)

 block_8_add (Add)              (None, 14, 14, 64)   0
['block_7_add[0][0]',
'block_8_project_BN[0][0]']

 block_9_expand (Conv2D)        (None, 14, 14, 384)  24576
['block_8_add[0][0]']

 block_9_expand_BN (BatchNormal  (None, 14, 14, 384)  1536
['block_9_expand[0][0]']
 ization)

 block_9_expand_relu (ReLU)     (None, 14, 14, 384)  0
['block_9_expand_BN[0][0]']

 block_9_depthwise (DepthwiseCo  (None, 14, 14, 384)  3456
['block_9_expand_relu[0][0]']
 nv2D)

 block_9_depthwise_BN (BatchNor  (None, 14, 14, 384)  1536
['block_9_depthwise[0][0]']
 malization)

 block_9_depthwise_relu (ReLU)  (None, 14, 14, 384)  0
['block_9_depthwise_BN[0][0]']

 block_9_project (Conv2D)       (None, 14, 14, 64)   24576
['block_9_depthwise_relu[0][0]']

 block_9_project_BN (BatchNorma  (None, 14, 14, 64)  256
['block_9_project[0][0]']
 lization)

 block_9_add (Add)              (None, 14, 14, 64)   0
['block_8_add[0][0]',
'block_9_project_BN[0][0]']

 block_10_expand (Conv2D)       (None, 14, 14, 384)  24576
['block_9_add[0][0]']

 block_10_expand_BN (BatchNorma  (None, 14, 14, 384)  1536
['block_10_expand[0][0]']
 lization)

 block_10_expand_relu (ReLU)    (None, 14, 14, 384)  0
['block_10_expand_BN[0][0]']
```

```
 block_10_depthwise (DepthwiseC  (None, 14, 14, 384)  3456
['block_10_expand_relu[0][0]']
 onv2D)

 block_10_depthwise_BN (BatchNo  (None, 14, 14, 384)  1536
['block_10_depthwise[0][0]']
 rmalization)

 block_10_depthwise_relu (ReLU)  (None, 14, 14, 384)  0
['block_10_depthwise_BN[0][0]']

 block_10_project (Conv2D)      (None, 14, 14, 96)   36864
['block_10_depthwise_relu[0][0]']

 block_10_project_BN (BatchNorm  (None, 14, 14, 96)   384
['block_10_project[0][0]']
 alization)

 block_11_expand (Conv2D)       (None, 14, 14, 576)  55296
['block_10_project_BN[0][0]']

 block_11_expand_BN (BatchNorma  (None, 14, 14, 576)  2304
['block_11_expand[0][0]']
 lization)

 block_11_expand_relu (ReLU)    (None, 14, 14, 576)  0
['block_11_expand_BN[0][0]']

 block_11_depthwise (DepthwiseC  (None, 14, 14, 576)  5184
['block_11_expand_relu[0][0]']
 onv2D)

 block_11_depthwise_BN (BatchNo  (None, 14, 14, 576)  2304
['block_11_depthwise[0][0]']
 rmalization)

 block_11_depthwise_relu (ReLU)  (None, 14, 14, 576)  0
['block_11_depthwise_BN[0][0]']

 block_11_project (Conv2D)      (None, 14, 14, 96)   55296
['block_11_depthwise_relu[0][0]']

 block_11_project_BN (BatchNorm  (None, 14, 14, 96)   384
['block_11_project[0][0]']
 alization)

 block_11_add (Add)             (None, 14, 14, 96)   0
```

```
['block_10_project_BN[0][0]',
'block_11_project_BN[0][0]']

 block_12_expand (Conv2D)       (None, 14, 14, 576)  55296
['block_11_add[0][0]']

 block_12_expand_BN (BatchNorma  (None, 14, 14, 576)  2304
['block_12_expand[0][0]']
 lization)

 block_12_expand_relu (ReLU)    (None, 14, 14, 576)  0
['block_12_expand_BN[0][0]']

 block_12_depthwise (DepthwiseC  (None, 14, 14, 576)  5184
['block_12_expand_relu[0][0]']
 onv2D)

 block_12_depthwise_BN (BatchNo  (None, 14, 14, 576)  2304
['block_12_depthwise[0][0]']
 rmalization)

 block_12_depthwise_relu (ReLU)  (None, 14, 14, 576)  0
['block_12_depthwise_BN[0][0]']

 block_12_project (Conv2D)      (None, 14, 14, 96)   55296
['block_12_depthwise_relu[0][0]']

 block_12_project_BN (BatchNorm  (None, 14, 14, 96)   384
['block_12_project[0][0]']
 alization)

 block_12_add (Add)             (None, 14, 14, 96)   0
['block_11_add[0][0]',
'block_12_project_BN[0][0]']

 block_13_expand (Conv2D)       (None, 14, 14, 576)  55296
['block_12_add[0][0]']

 block_13_expand_BN (BatchNorma  (None, 14, 14, 576)  2304
['block_13_expand[0][0]']
 lization)

 block_13_expand_relu (ReLU)    (None, 14, 14, 576)  0
['block_13_expand_BN[0][0]']

 block_13_pad (ZeroPadding2D)   (None, 15, 15, 576)  0
['block_13_expand_relu[0][0]']
```

```
 block_13_depthwise (DepthwiseC  (None, 7, 7, 576)   5184
['block_13_pad[0][0]']
 onv2D)

 block_13_depthwise_BN (BatchNo  (None, 7, 7, 576)   2304
['block_13_depthwise[0][0]']
 rmalization)

 block_13_depthwise_relu (ReLU)  (None, 7, 7, 576)   0
['block_13_depthwise_BN[0][0]']

 block_13_project (Conv2D)      (None, 7, 7, 160)    92160
['block_13_depthwise_relu[0][0]']

 block_13_project_BN (BatchNorm  (None, 7, 7, 160)   640
['block_13_project[0][0]']
 alization)

 block_14_expand (Conv2D)       (None, 7, 7, 960)    153600
['block_13_project_BN[0][0]']

 block_14_expand_BN (BatchNorma  (None, 7, 7, 960)   3840
['block_14_expand[0][0]']
 lization)

 block_14_expand_relu (ReLU)    (None, 7, 7, 960)    0
['block_14_expand_BN[0][0]']

 block_14_depthwise (DepthwiseC  (None, 7, 7, 960)   8640
['block_14_expand_relu[0][0]']
 onv2D)

 block_14_depthwise_BN (BatchNo  (None, 7, 7, 960)   3840
['block_14_depthwise[0][0]']
 rmalization)

 block_14_depthwise_relu (ReLU)  (None, 7, 7, 960)   0
['block_14_depthwise_BN[0][0]']

 block_14_project (Conv2D)      (None, 7, 7, 160)    153600
['block_14_depthwise_relu[0][0]']

 block_14_project_BN (BatchNorm  (None, 7, 7, 160)   640
['block_14_project[0][0]']
 alization)

 block_14_add (Add)             (None, 7, 7, 160)    0
['block_13_project_BN[0][0]',
'block_14_project_BN[0][0]']

 block_15_expand (Conv2D)       (None, 7, 7, 960)    153600
['block_14_add[0][0]']

 block_15_expand_BN (BatchNorma  (None, 7, 7, 960)   3840
['block_15_expand[0][0]']
 lization)

 block_15_expand_relu (ReLU)    (None, 7, 7, 960)    0
['block_15_expand_BN[0][0]']

 block_15_depthwise (DepthwiseC  (None, 7, 7, 960)   8640
['block_15_expand_relu[0][0]']
 onv2D)

 block_15_depthwise_BN (BatchNo  (None, 7, 7, 960)   3840
['block_15_depthwise[0][0]']
 rmalization)

 block_15_depthwise_relu (ReLU)  (None, 7, 7, 960)   0
['block_15_depthwise_BN[0][0]']

 block_15_project (Conv2D)      (None, 7, 7, 160)    153600
['block_15_depthwise_relu[0][0]']

 block_15_project_BN (BatchNorm  (None, 7, 7, 160)   640
['block_15_project[0][0]']
 alization)

 block_15_add (Add)             (None, 7, 7, 160)    0
['block_14_add[0][0]',
'block_15_project_BN[0][0]']

 block_16_expand (Conv2D)       (None, 7, 7, 960)    153600
['block_15_add[0][0]']

 block_16_expand_BN (BatchNorma  (None, 7, 7, 960)   3840
['block_16_expand[0][0]']
 lization)

 block_16_expand_relu (ReLU)    (None, 7, 7, 960)    0
['block_16_expand_BN[0][0]']

 block_16_depthwise (DepthwiseC  (None, 7, 7, 960)   8640
['block_16_expand_relu[0][0]']
 onv2D)
 block_16_depthwise_BN (BatchNo  (None, 7, 7, 960)   3840
['block_16_depthwise[0][0]']
 rmalization)

 block_16_depthwise_relu (ReLU)  (None, 7, 7, 960)   0
['block_16_depthwise_BN[0][0]']

 block_16_project (Conv2D)      (None, 7, 7, 320)    307200
['block_16_depthwise_relu[0][0]']

 block_16_project_BN (BatchNorm  (None, 7, 7, 320)   1280
['block_16_project[0][0]']
 alization)

 Conv_1 (Conv2D)                (None, 7, 7, 1280)   409600
['block_16_project_BN[0][0]']

 Conv_1_bn (BatchNormalization)  (None, 7, 7, 1280)  5120
['Conv_1[0][0]']

 out_relu (ReLU)                (None, 7, 7, 1280)   0
['Conv_1_bn[0][0]']

================================================================================
==================
Total params: 2,257,984
Trainable params: 2,223,872
Non-trainable params: 34,112
--------------------------------------------------------------------------------
------------------
```

[12]:

```python
# 5. Replace Fully Connected Layers

# Create Model Structure
num_classes =  len(list(train_gen.class_indices.keys())) # to define number of␣
 ↪classes in dense layer


model = keras.Sequential(
    [
        base_model,
        layers.Flatten(),
        layers.Dense(1024, activation="relu"),
        layers.Dense(512, activation="relu"),
        layers.Dense(num_classes, activation="softmax"),
    ]
)
```

```
#model.compile(Adamax(learning_rate= 0.001), loss= 'categorical_crossentropy',␣
 ↪metrics= ['accuracy'])
```

[13]:
```
# 6. Total Parameters

total_params = model.count_params()
print(f"Total Parameters: {total_params}")
```

```
Total Parameters: 67011140
```

[14]:
```
# 7. Print Architecture
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 mobilenetv2_1.00_224 (Funct  (None, 7, 7, 1280)       2257984
 ional)

 flatten (Flatten)           (None, 62720)             0

 dense (Dense)               (None, 1024)              64226304

 dense_1 (Dense)             (None, 512)               524800

 dense_2 (Dense)             (None, 4)                 2052

=================================================================
Total params: 67,011,140
Trainable params: 66,977,028
Non-trainable params: 34,112
_________________________________________________________________
```

[15]:
```
batch_size = 64   #batch size for training
epochs = 20  # number of all epochs in training
patience = 3   #number of epochs to wait to adjust lr if monitored value does␣
 ↪not improve
stop_patience = 10   # number of epochs to wait before stopping training if␣
 ↪monitored value does not improve
threshold = 0.9   # if train accuracy is < threshold adjust monitor accuracy,␣
 ↪else monitor validation loss
factor = 0.5   # factor to reduce lr by
ask_epoch = 50   # number of epochs to run before asking if you want to halt␣
 ↪training
```

```
batches = int(np.ceil(len(train_gen.labels) / batch_size))    # number of␣
 ↪training batch to run per epoch
print(f'Number of batches in training: {batches*64}')
```

```
Number of batches in training: 16960
```

[16]:

```
# 8. Retrain with Transfer Learning

def train_model(model):

    # Compile and train model
    model.compile(
        optimizer=keras.optimizers.Adam(learning_rate=0.001),
        loss="categorical_crossentropy",
        metrics=["accuracy"],
    )
    print(model.summary())
    callbacks = [MyCallback(model= model, patience= patience, stop_patience=␣
 ↪stop_patience, threshold= threshold,
            factor= factor, batches= batches, epochs= epochs, ask_epoch=␣
 ↪ask_epoch )]

    history = model.fit(
       x= train_gen, epochs= epochs, batch_size= batch_size, callbacks=␣
 ↪callbacks,
                   validation_data= valid_gen, validation_steps= None, shuffle=␣
 ↪False, verbose=0
     )
    return history

# define a function to set number of trainable layers
def set_trainable_layers(trainable_layers):
    # Freeze base model layers
    tempBASEMODEL = getBaseModel()
    for layer in tempBASEMODEL.layers:
        layer.trainable = False

    # Unfreeze specific layers
    if trainable_layers == 4:
        for layer in tempBASEMODEL.layers[-3:]:
            layer.trainable = True
            print(f"Layer {layer.name} is trainable")
    elif trainable_layers == 5:
        for layer in tempBASEMODEL.layers[-5:]:
            layer.trainable = True
            print(f"Layer {layer.name} is trainable")
    model = keras.Sequential(
    [
        tempBASEMODEL,
        layers.Flatten(),
        layers.Dense(1024, activation="relu"),
        layers.Dense(512, activation="relu"),
        layers.Dense(num_classes, activation="softmax"),
    ]
    )

    return model

model1 = set_trainable_layers( trainable_layers=3) # Last 3 FC layers
model2 = set_trainable_layers( trainable_layers=4) # 1 Conv + 3 FC layers
model3 = set_trainable_layers( trainable_layers=5)  # 2 Conv + 3 FC layers

# Train with different sets of trainable layers
history_3 = train_model(model1)  # Last 3 FC layers
history_4 = train_model(model2)  # 1 Conv + 3 FC layers
history_5 = train_model(model3)  # 2 Conv + 3 FC layers

# 9. Evaluation and Comparison

# (Code for calculating accuracy, recall, precision, F1-score, sensitivity,
# and plotting convergence curves would go here)
```

```
Layer Conv_1 is trainable
Layer Conv_1_bn is trainable
Layer out_relu is trainable
Layer block_16_project is trainable
Layer block_16_project_BN is trainable
Layer Conv_1 is trainable
Layer Conv_1_bn is trainable
Layer out_relu is trainable
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 mobilenetv2_1.00_224 (Funct  (None, 7, 7, 1280)       2257984
 ional)

 flatten_1 (Flatten)         (None, 62720)             0

 dense_3 (Dense)             (None, 1024)              64226304

 dense_4 (Dense)             (None, 512)               524800
 dense_5 (Dense)             (None, 4)                 2052

=================================================================
Total params: 67,011,140
Trainable params: 64,753,156
Non-trainable params: 2,257,984
_________________________________________________________________
None
Do you want model asks you to halt the training [y/n] ?
 Epoch     Loss   Accuracy  V_loss    V_acc     LR     Next LR  Monitor  %
Improv  Duration
 1 /20     1.178   76.695   0.49289  81.994   0.00100  0.00100  accuracy
0.00    162.75
 2 /20     0.438   83.942   0.39952  84.972   0.00100  0.00100  accuracy
9.45    64.17
 3 /20     0.374   85.920   0.48032  83.270   0.00100  0.00100  accuracy
2.36    64.65
 4 /20     0.326   87.863   0.39263  85.917   0.00100  0.00100  accuracy
2.26    62.55
 5 /20     0.293   89.452   0.41083  85.350   0.00100  0.00100  accuracy
1.81    62.71
 6 /20     0.278   89.659   0.36501  87.004   0.00100  0.00100  accuracy
0.23    64.04
 7 /20     0.253   90.621   0.39482  86.862   0.00100  0.00100  val_loss
-8.17    61.80
 8 /20     0.240   91.206   0.35082  86.862   0.00100  0.00100  val_loss
3.89    64.74
 9 /20     0.213   92.251   0.41785  85.775   0.00100  0.00100  val_loss
-19.11   62.67
10 /20     0.210   92.245   0.47447  85.113   0.00100  0.00100  val_loss
-35.25   62.54
11 /20     0.196   92.623   0.36808  86.437   0.00100  0.00050  val_loss
-4.92    62.75
12 /20     0.143   94.673   0.38946  87.760   0.00050  0.00050  val_loss
-11.02   62.33
13 /20     0.131   95.263   0.38550  87.807   0.00050  0.00050  val_loss
-9.89    63.42
14 /20     0.122   95.659   0.41437  87.098   0.00050  0.00025  val_loss
-18.11   61.71
15 /20     0.095   96.781   0.42801  88.138   0.00025  0.00025  val_loss
-22.00   61.77
16 /20     0.093   96.687   0.47304  87.335   0.00025  0.00025  val_loss
-34.84   62.28
17 /20     0.084   97.000   0.46536  88.138   0.00025  0.00013  val_loss
-32.65   61.32
18 /20     0.069   97.632   0.51962  86.815   0.00013  0.00013  val_loss
-48.12   66.69
19 /20     0.062   97.868   0.51418  87.713   0.00013  0.00013  val_loss
```

```
-46.57   62.70
20 /20     0.065   97.903   0.52973  87.949   0.00013  0.00006  val_loss
-51.00   62.75
training elapsed time was 0.0 hours, 22.0 minutes, 43.27 seconds)
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 mobilenetv2_1.00_224 (Funct  (None, 7, 7, 1280)       2257984
 ional)

 flatten_2 (Flatten)         (None, 62720)             0

 dense_6 (Dense)             (None, 1024)              64226304

 dense_7 (Dense)             (None, 512)               524800

 dense_8 (Dense)             (None, 4)                 2052

=================================================================
Total params: 67,011,140
Trainable params: 65,165,316
Non-trainable params: 1,845,824
_________________________________________________________________
None
Do you want model asks you to halt the training [y/n] ?
 Epoch     Loss   Accuracy  V_loss    V_acc     LR     Next LR  Monitor  %
Improv  Duration
 1 /20     0.994   77.185   0.59761  77.977   0.00100  0.00100  accuracy
0.00    67.89
 2 /20     0.448   83.593   0.76103  78.403   0.00100  0.00100  accuracy
8.30    61.26
 3 /20     0.371   86.257   0.67452  80.340   0.00100  0.00100  accuracy
3.19    62.03
 4 /20     0.328   87.657   0.48797  85.681   0.00100  0.00100  accuracy
1.62    60.86
 5 /20     0.305   88.696   0.57162  81.758   0.00100  0.00100  accuracy
1.19    61.87
 6 /20     0.295   88.962   0.40935  85.255   0.00100  0.00100  accuracy
0.30    60.80
 7 /20     0.272   90.043   0.40576  86.011   0.00100  0.00100  val_loss
0.88    63.08
 8 /20     0.258   90.149   0.58749  83.365   0.00100  0.00100  val_loss
-44.79   61.11
 9 /20     0.232   91.389   0.38237  86.862   0.00100  0.00100  val_loss
5.77    62.90
10 /20     0.232   91.425   0.40723  87.760   0.00100  0.00100  val_loss
-6.50    64.64
```

```
11 /20     0.226   91.814   0.60903  84.783   0.00100  0.00100  val_loss
-59.28   61.64
12 /20     0.199   93.043   0.46314  84.074   0.00100  0.00050  val_loss
-21.12   61.55
13 /20     0.141   94.933   0.38231  88.658   0.00050  0.00050  val_loss
0.02    61.21
14 /20     0.121   95.576   0.37215  89.083   0.00050  0.00050  val_loss
2.66    61.62
15 /20     0.104   96.368   0.41466  88.422   0.00050  0.00050  val_loss
-11.42   61.64
16 /20     0.093   96.799   0.37416  88.894   0.00050  0.00050  val_loss
-0.54    62.00
17 /20     0.087   96.905   0.45120  88.422   0.00050  0.00025  val_loss
-21.24   62.30
18 /20     0.056   98.210   0.57556  88.705   0.00025  0.00025  val_loss
-54.66   64.13
19 /20     0.049   98.435   0.54046  88.800   0.00025  0.00025  val_loss
-45.22   63.76
20 /20     0.038   98.630   0.58730  89.319   0.00025  0.00013  val_loss
-57.81   61.23
training elapsed time was 0.0 hours, 20.0 minutes, 51.50 seconds)
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 mobilenetv2_1.00_224 (Funct  (None, 7, 7, 1280)       2257984   
 ional)                                                          
                                                                 
 flatten_3 (Flatten)         (None, 62720)             0         
                                                                 
 dense_9 (Dense)             (None, 1024)              64226304  
                                                                 
 dense_10 (Dense)            (None, 512)               524800    
                                                                 
 dense_11 (Dense)            (None, 4)                 2052      
                                                                 
=================================================================
Total params: 67,011,140
Trainable params: 65,473,156
Non-trainable params: 1,537,984
_________________________________________________________________
None
Do you want model asks you to halt the training [y/n] ?
 Epoch     Loss   Accuracy  V_loss    V_acc     LR     Next LR  Monitor  %
Improv  Duration
 1 /20     0.945   76.595  10.97996  30.624   0.00100  0.00100  accuracy
0.00    71.14
 2 /20     0.458   83.404   5.11623  42.580   0.00100  0.00100  accuracy
```

```
8.89    66.15
 3 /20     0.375   86.168   5.98740  34.924   0.00100  0.00100  accuracy
3.31    66.23
 4 /20     0.340   87.320   4.25912  37.004   0.00100  0.00100  accuracy
1.34    62.41
 5 /20     0.334   87.609   1.93340  55.766   0.00100  0.00100  accuracy
0.33    62.56
 6 /20     0.306   88.661   6.28908  25.142   0.00100  0.00100  accuracy
1.20    62.37
 7 /20     0.301   89.003   2.98812  53.166   0.00100  0.00100  accuracy
0.39    62.12
 8 /20     0.267   90.019   4.98894  42.864   0.00100  0.00100  val_loss
-158.04   62.31
 9 /20     0.262   90.385   2.49987  63.469   0.00100  0.00100  val_loss
-29.30   63.33
10 /20     0.250   90.804   3.20290  50.095   0.00100  0.00050  val_loss
-65.66   63.12
11 /20     0.190   92.919   0.79539  81.474   0.00050  0.00050  val_loss
58.86    62.78
12 /20     0.167   93.698   0.60830  85.208   0.00050  0.00050  val_loss
23.52    61.97
13 /20     0.156   94.200   0.85854  81.049   0.00050  0.00050  val_loss
-41.14   60.92
14 /20     0.142   94.850   0.68134  81.853   0.00050  0.00050  val_loss
-12.01   62.07
15 /20     0.137   95.027   0.53209  85.444   0.00050  0.00050  val_loss
12.53    63.57
16 /20     0.122   95.582   1.02781  80.246   0.00050  0.00050  val_loss
-93.16   63.31
17 /20     0.106   96.208   0.57790  86.153   0.00050  0.00050  val_loss
-8.61    61.42
18 /20     0.108   96.244   0.88770  82.278   0.00050  0.00025  val_loss
-66.83   62.24
19 /20     0.072   97.525   0.66038  86.200   0.00025  0.00025  val_loss
-24.11   60.91
20 /20     0.053   98.228   0.58933  86.626   0.00025  0.00025  val_loss
-10.76   62.11
training elapsed time was 0.0 hours, 21.0 minutes, 7.08 seconds)
```

[25]:

```
model1.save(f'saved_model/Model{1}.h5', save_format="h5")
model2.save(f'saved_model/Model{2}.h5', save_format="h5")
model3.save(f'saved_model/Model{3}.h5', save_format="h5")
```

[27]:

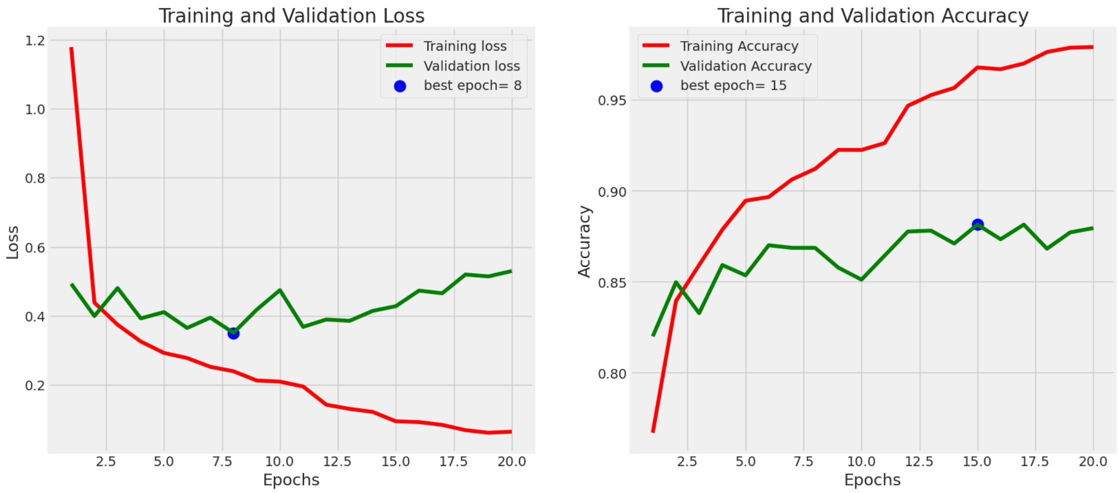
```
!ls /kaggle/working/saved_model
from IPython.display import FileLink
display(FileLink("saved_model/Model1.h5"))
display(FileLink("saved_model/Model2.h5"))
display(FileLink("saved_model/Model3.h5"))
```

```
Model1  Model1.h5  Model2  Model2.h5  Model3  Model3.h5

/kaggle/working/saved_model/Model1.h5

/kaggle/working/saved_model/Model2.h5

/kaggle/working/saved_model/Model3.h5
```

[19]:

```
history = [history_3, history_4, history_5 ]
i = 1
for his in history:
    print(f'Model_{i}')
    plot_training(his)
    i+=1
```

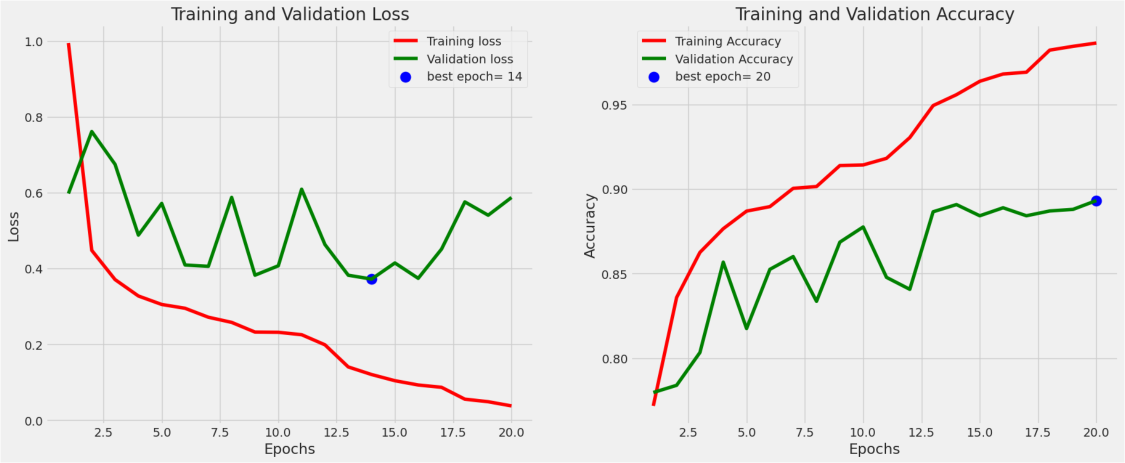
```
Model_1
```



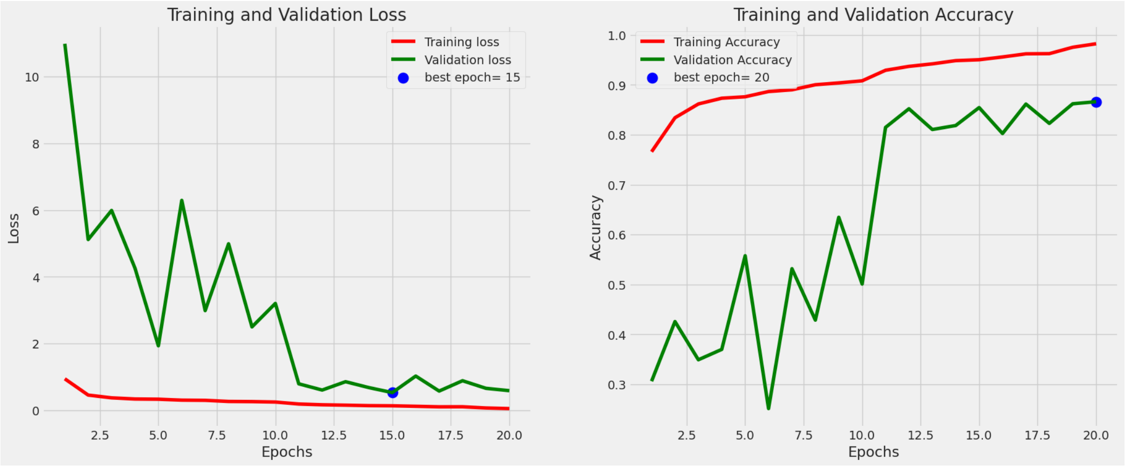
Training and Validation Loss

Training and Validation Accuracy

```
Model_2
```



Model_3



[20]:

```
models = [model1, model2, model3]
i =1
for model in models:
    ts_length = len(test_df)
    test_batch_size = test_batch_size = max(sorted([ts_length // n for n in␣
 ↪range(1, ts_length + 1) if ts_length%n == 0 and ts_length/n <= 80]))
    test_steps = ts_length // test_batch_size

    train_score = model.evaluate(train_gen, steps= test_steps, verbose= 1)
    valid_score = model.evaluate(valid_gen, steps= test_steps, verbose= 1)
    test_score = model.evaluate(test_gen, steps= test_steps, verbose= 1)
    print(f"==========model_{i}==============" )
    print("Train Loss: ", train_score[0])
    print("Train Accuracy: ", train_score[1])
    print('-' * 20)
    print("Validation Loss: ", valid_score[0])
    print("Validation Accuracy: ", valid_score[1])
    print('-' * 20)
    print("Test Loss: ", test_score[0])
    print("Test Accuracy: ", test_score[1])
    print("=========================" )
    i+=1
```

```
29/29 [==============================] - 2s 54ms/step - loss: 0.2619 - accuracy:
0.9052
29/29 [==============================] - 2s 50ms/step - loss: 0.3450 - accuracy:
0.8621
29/29 [==============================] - 20s 678ms/step - loss: 0.3419 -
accuracy: 0.8781
===========model_1==============
Train Loss:  0.2619060277938843
Train Accuracy:  0.9051724076271057
--------------------
Validation Loss:  0.3450435996055603
Validation Accuracy:  0.8620689511299133
--------------------
Test Loss:  0.3418533205986023
Test Accuracy:  0.8781294226646423
=========================
29/29 [==============================] - 2s 54ms/step - loss: 0.0881 - accuracy:
0.9698
29/29 [==============================] - 2s 52ms/step - loss: 0.4603 - accuracy:
0.8728
29/29 [==============================] - 8s 282ms/step - loss: 0.3341 -
accuracy: 0.9017
===========model_2==============
Train Loss:  0.08808482438325882
Train Accuracy:  0.9698275923728943
--------------------
Validation Loss:  0.46028727293014526
Validation Accuracy:  0.8728448152542114
--------------------
Test Loss:  0.334089070558548
Test Accuracy:  0.9017477631568909
=========================
29/29 [==============================] - 2s 54ms/step - loss: 0.2328 - accuracy:
0.9203
29/29 [==============================] - 1s 47ms/step - loss: 0.5803 - accuracy:
0.8384
29/29 [==============================] - 8s 267ms/step - loss: 0.5104 -
```

```
accuracy: 0.8715
===========model_3==============
Train Loss:  0.23283062875270844
Train Accuracy:  0.920258641242981
--------------------
Validation Loss:  0.580291748046875
Validation Accuracy:  0.8383620977401733
--------------------
Test Loss:  0.510432243347168
Test Accuracy:  0.8715162873268127
=========================
```

[28]:

```
Preds = {}
i = 1
for model in models:
    preds = model.predict_generator(test_gen)
    y_pred = np.argmax(preds, axis=1)
    Preds[f"Model{i}"] = y_pred
    i+=1

print(Preds)
```

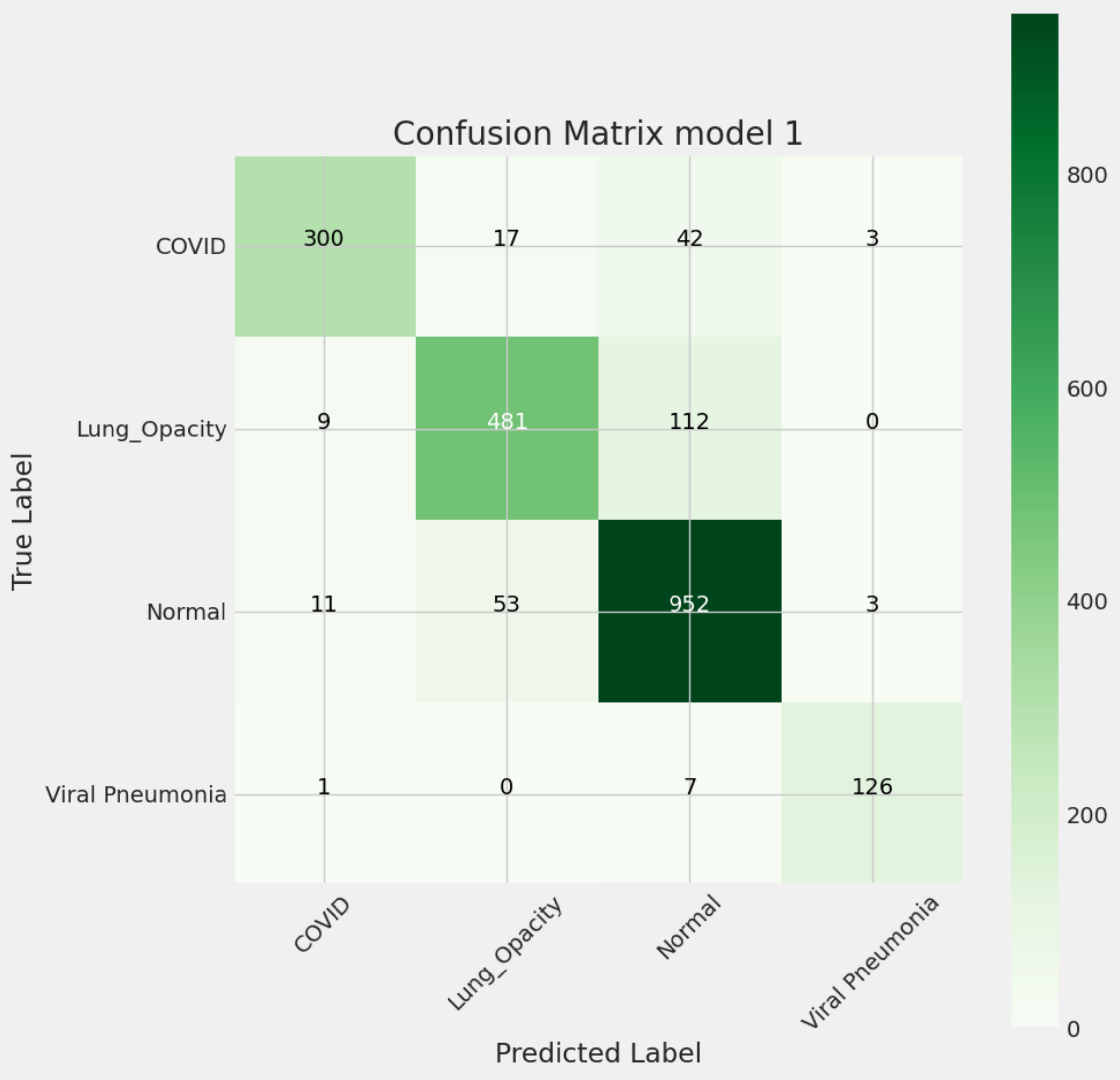
```
{'Model1': array([1, 1, 1, …, 2, 2, 2]), 'Model2': array([1, 1, 1, …, 2, 2,
2]), 'Model3': array([1, 1, 1, …, 2, 2, 2])}
```

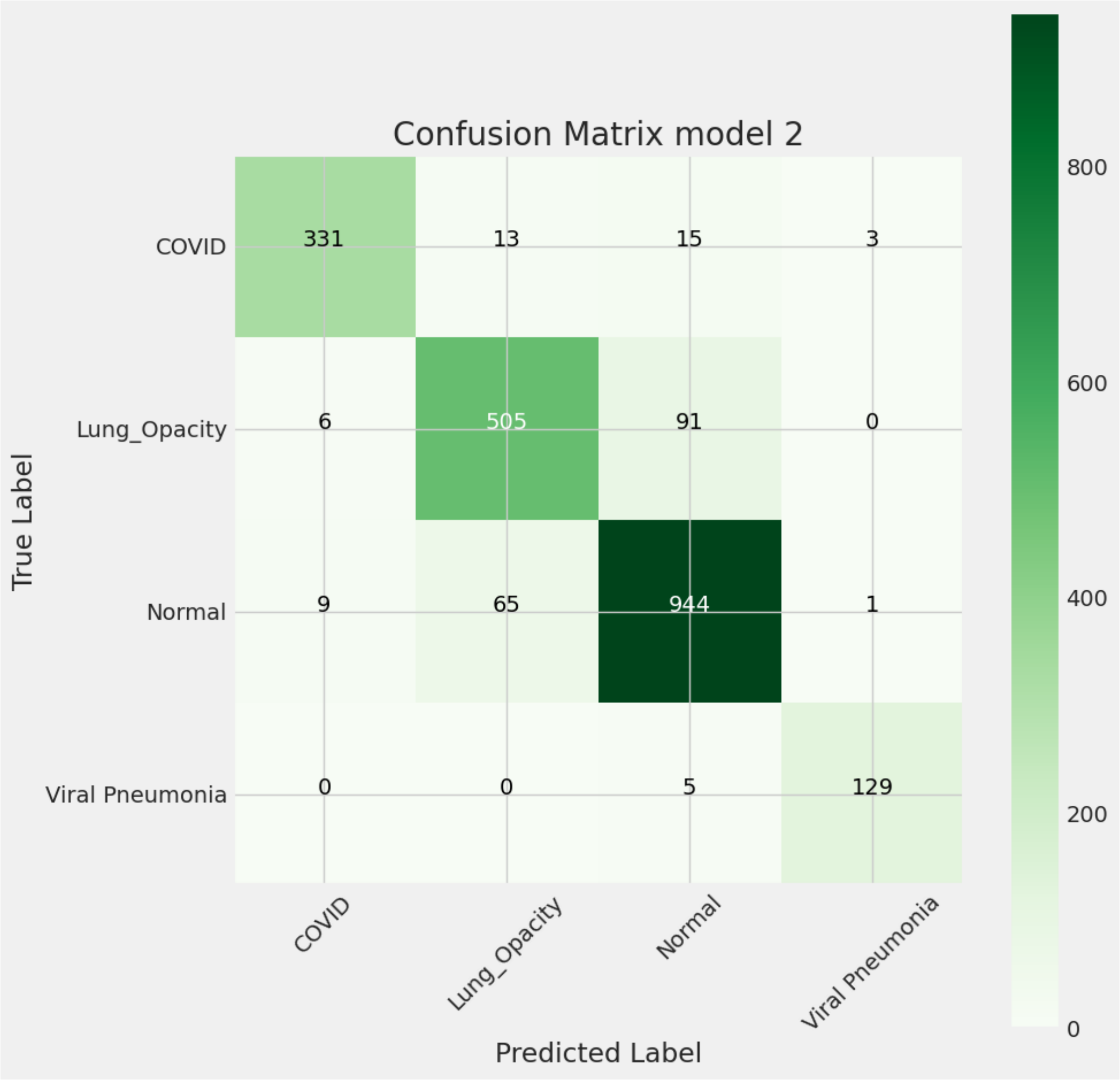
[36]:

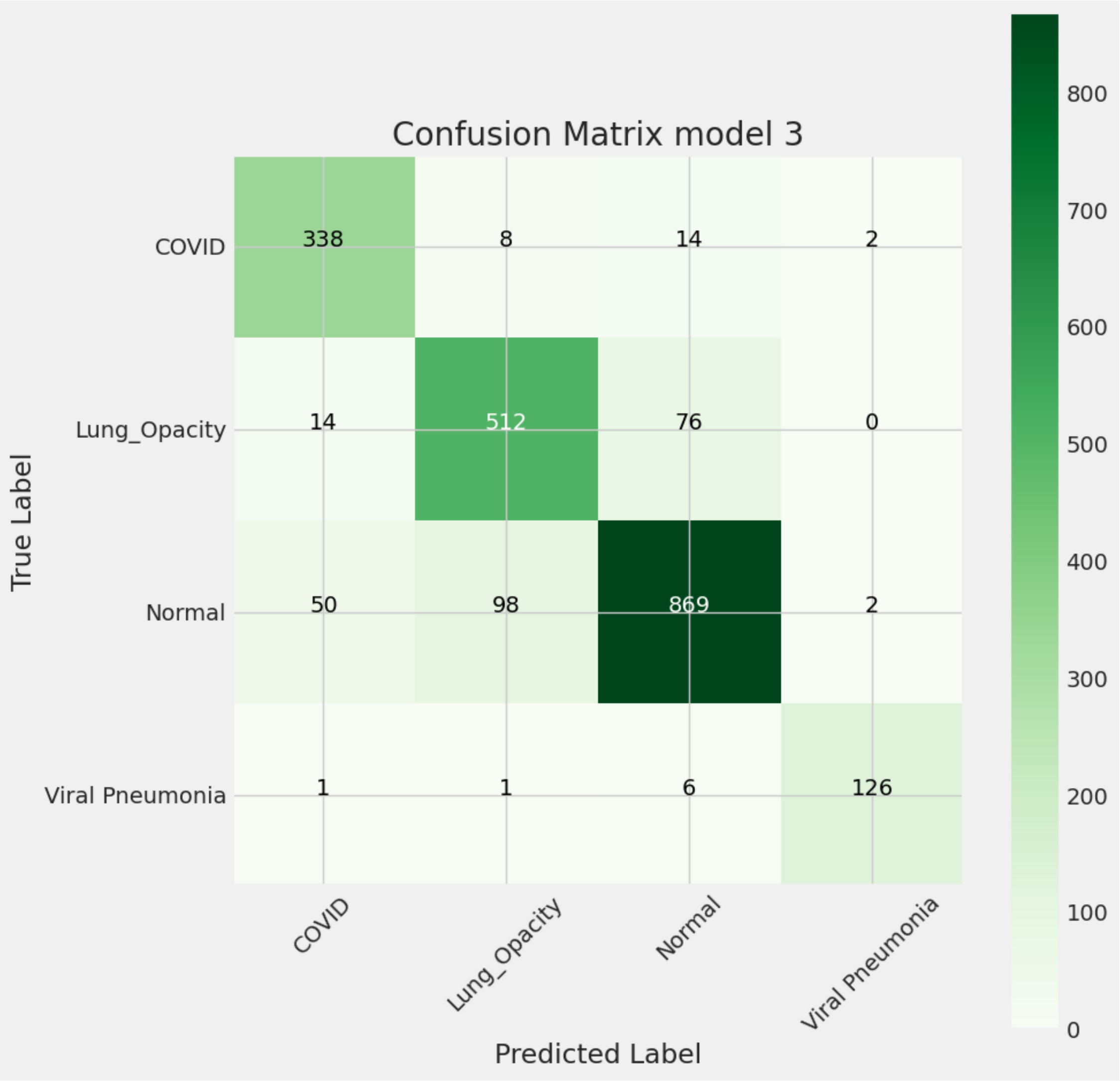
```
i =1
for model in models:
    print(f"===============Model {i}===============")
    y_pred = Preds[f"Model{i}"]
    g_dict = test_gen.class_indices
    classes = list(g_dict.keys())

    # Confusion matrix
    cm = confusion_matrix(test_gen.classes, y_pred)
    plot_confusion_matrix(cm= cm, classes= classes, title = f'Confusion Matrix␣
 ↪model {i}')

    # Classification report
    print(classification_report(test_gen.classes, y_pred, target_names=␣
 ↪classes))
    i+=1
    print("===============================")
```

```
===============Model 1===============
Confusion Matrix, Without Normalization
[[300  17  42   3]
 [  9 481 112   0]
 [ 11  53 952   3]
 [  1   0   7 126]]
                 precision    recall  f1-score   support

          COVID       0.93      0.83      0.88       362
   Lung_Opacity       0.87      0.80      0.83       602
         Normal       0.86      0.93      0.89      1019
Viral Pneumonia       0.95      0.94      0.95       134

       accuracy                           0.88      2117
      macro avg       0.90      0.88      0.89      2117
   weighted avg       0.88      0.88      0.88      2117

===============================
===============Model 2===============
Confusion Matrix, Without Normalization
[[331  13  15   3]
 [  6 505  91   0]
 [  9  65 944   1]
 [  0   0   5 129]]
                 precision    recall  f1-score   support

          COVID       0.96      0.91      0.94       362
   Lung_Opacity       0.87      0.84      0.85       602
         Normal       0.89      0.93      0.91      1019
Viral Pneumonia       0.97      0.96      0.97       134

       accuracy                           0.90      2117
      macro avg       0.92      0.91      0.92      2117
   weighted avg       0.90      0.90      0.90      2117

===============================
===============Model 3===============
Confusion Matrix, Without Normalization
[[338   8  14   2]
 [ 14 512  76   0]
 [ 50  98 869   2]
 [  1   1   6 126]]
                 precision    recall  f1-score   support

          COVID       0.84      0.93      0.88       362
   Lung_Opacity       0.83      0.85      0.84       602
         Normal       0.90      0.85      0.88      1019
Viral Pneumonia       0.97      0.94      0.95       134

       accuracy                           0.87      2117
      macro avg       0.88      0.89      0.89      2117
   weighted avg       0.87      0.87      0.87      2117
```

```
================================
```


Confusion Matrix model 1
True Label
Predicted Label
COVID
Lung_Opacity
Normal
Viral Pneumonia
300
17
42
3
9
481
112
0
11
53
952
3
1
0
7
126
COVID
Lung_Opacity
Normal
Viral Pneumonia
800
600
400
200
0


Confusion Matrix model 2
True Label
Predicted Label
COVID
Lung_Opacity
Normal
Viral Pneumonia
331
13
15
3
6
505
91
0
9
65
944
1
0
0
5
129
COVID
Lung_Opacity
Normal
Viral Pneumonia
800
600
400
200
0


Confusion Matrix model 3
True Label
COVID
Lung_Opacity
Normal
Viral Pneumonia
338
8
14
2
14
512
76
0
50
98
869
2
1
1
6
126
Predicted Label
COVID
Lung_Opacity
Normal
Viral Pneumonia
800
700
600
500
400
300
200
100
0