from \_\_future\_\_ import absolute\_import, division, print\_function

import tensorflow as tf

from tensorflow import keras

import functions as f

print("TensorFlow version is ", tf.\_\_version\_\_)

import matplotlib.pyplot as plt

import build\_model as build

import pandas as pd

base\_dir = f.get\_base\_dir()

print(f.get\_base\_dir())

train\_dir = f.get\_train\_dir()

print(f.get\_train\_dir())

val\_dir = f.get\_val\_dir()

print(f.get\_val\_dir())

#get subdirectories

train\_subdir = f.get\_subdirectories(train\_dir)

val\_subdir = f.get\_subdirectories(val\_dir)

n = train\_subdir.\_\_len\_\_()

#create image data generator with image augmentation

image\_size = 160 # All images will be resized to 160x160

batch\_size = 32

IMG\_SHAPE = (image\_size, image\_size, 3)

classes = []

classes.append('damage')

classes.append('whole')

print('{} vs {}'.format(classes[0], classes[1]))

# Rescale all images by 1./255 and apply image augmentation

train\_datagen = keras.preprocessing.image.ImageDataGenerator(

rescale=1./255)

validation\_datagen = keras.preprocessing.image.ImageDataGenerator(rescale=1./255)

# Flow training images in batches of 20 using train\_datagen generator

train\_generator = train\_datagen.flow\_from\_directory(

train\_dir, # Source directory for the training images

classes={classes[0]: 0,

classes[1]: 1},

target\_size=(image\_size, image\_size),

batch\_size=batch\_size,

# Since we use binary\_crossentropy loss, we need binary labels

class\_mode='binary')

# Flow validation images in batches of 20 using test\_datagen generator

validation\_generator = validation\_datagen.flow\_from\_directory(

val\_dir, # Source directory for the validation images

classes={classes[0]:0,

classes[1]:1},

target\_size=(image\_size, image\_size),

batch\_size=batch\_size,

class\_mode='binary')

# Create the base model from the pre-trained model MobileNet V2

base\_model = tf.keras.applications.MobileNetV2(input\_shape=IMG\_SHAPE,

include\_top=False,

weights='imagenet')

base\_model.trainable = False

# classification head

model = build.simple\_binary\_model(base\_model)

# compilation of the model

model.compile(optimizer=tf.keras.optimizers.RMSprop(lr=0.0001),

loss='binary\_crossentropy',

metrics=['accuracy'])

# Training

epochs = 20

steps\_per\_epoch = train\_generator.n // batch\_size

validation\_steps = validation\_generator.n // batch\_size

history = model.fit\_generator(train\_generator,

steps\_per\_epoch=steps\_per\_epoch,

epochs=epochs,

workers=4,

validation\_data=validation\_generator,

validation\_steps=validation\_steps,

verbose=1)

# accuracy and loss graph

acc = history.history['acc']

val\_acc = history.history['val\_acc']

loss = history.history['loss']

val\_loss = history.history['val\_loss']

# fine tuning

# unfreeze the top layers of the model

base\_model.trainable = True

# Layers in the base model

#print("Number of layers in the base model: ", len(base\_model.layers))

# Fine tune from this layer onwards

fine\_tune\_at = 100

# Freeze all the layers before the `fine\_tune\_at` layer

for layer in base\_model.layers[:fine\_tune\_at]:

layer.trainable = False

# recompiling the model

model.compile(loss='binary\_crossentropy',

optimizer=tf.keras.optimizers.RMSprop(lr=2e-5),

metrics=['accuracy'])

#model.summary()

#print('# of variables: {}'.format(len(model.trainable\_variables)))

# training

history\_fine = model.fit\_generator(train\_generator,

steps\_per\_epoch=steps\_per\_epoch,

epochs=epochs,

workers=4,

validation\_data=validation\_generator,

validation\_steps=validation\_steps,

verbose=1)

# learning curves

acc += history\_fine.history['acc']

val\_acc += history\_fine.history['val\_acc']

loss += history\_fine.history['loss']

val\_loss += history\_fine.history['val\_loss']

plt.figure(figsize=(8, 8))

plt.subplot(2, 1, 1)

plt.plot(acc, label='Training Accuracy')

plt.plot(val\_acc, label='Validation Accuracy')

plt.ylim([.3, 1])

plt.plot([epochs - 1, epochs - 1], plt.ylim(), label='Start Fine Tuning')

plt.legend(loc='lower right')

plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)

plt.plot(loss, label='Training Loss')

plt.plot(val\_loss, label='Validation Loss')

plt.ylim([0, 1])

plt.plot([epochs - 1, epochs - 1], plt.ylim(), label='Start Fine Tuning')

plt.legend(loc='upper right')

plt.title('Training and Validation Loss')

filename = 'out/' + classes[0] + '-vs-' + classes[1] + '\_binary\_test.png'

plt.savefig(filename)

pd.DataFrame(history.history['val\_acc']).to\_csv('accuracies/'+ classes[0] +'-vs-'+ classes[1] +'binary\_test.csv')

model\_path = 'models/' + classes[0] + 'v' + classes[1]

model.save\_weights(model\_path + 'model\_weights.h5')

with open(model\_path + 'model\_architecture.json', 'w') as f:

f.write(model.to\_json())