#Automated Port Operation

import os

import shutil

import splitfolders

import tensorflow as tf

from os import listdir

import matplotlib.pyplot as plt

import seaborn as sns

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.applications import MobileNet

from tensorflow.keras.layers import GlobalAveragePooling2D, Dense, Dropout, BatchNormalization, Conv2D, MaxPooling2D

from tensorflow.keras.models import Model, Sequential

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.callbacks import EarlyStopping

from keras.metrics import Recall, Precision

from sklearn.metrics import confusion\_matrix, ConfusionMatrixDisplay, classification\_report

import pandas as pd

import numpy as np

from sklearn.utils import class\_weight

import warnings

warnings.simplefilter(action='ignore')

#Listing input images

def get\_file\_ext(file\_name):

    split\_tup = os.path.splitext(file\_name)

    return split\_tup[1]

def is\_image\_file(file\_ext):

    if file\_ext == '.jpeg' or file\_ext == '.jpg' or \

        file\_ext == '.gif' or file\_ext == '.png':

        return True

    else:

        return False

def list\_image\_files(rootdir):

    images\_dirs = listdir(path=rootdir)

    image\_dir\_dict = {}

    i = 1

    print('-------------------------------------------------------------')

    print(f'Following sub-directories found in "{rootdir}" directory')

    for dirs in images\_dirs:

        print(f'{i}: {dirs}')

        i = i+1

    print('-------------------------------------------------------------')

    total\_image\_count = 0

    for dirs in images\_dirs:

        file\_list = listdir(path=rootdir + '/' + dirs)

        image\_files = [ file\_name for file\_name in file\_list \

                       if is\_image\_file(get\_file\_ext(file\_name)) ]

        image\_count = len(image\_files)

        total\_image\_count = total\_image\_count + image\_count

        image\_dir\_dict[dirs]=image\_count

    print(f'Total images found = {total\_image\_count}')

    for key in image\_dir\_dict.keys():

        dir\_image\_count = image\_dir\_dict[key]

        print(f'Directory "{key}" contains {dir\_image\_count} image files' + \

              f' = {round(dir\_image\_count/total\_image\_count,3):.3%}')

list\_image\_files('Automating\_Port\_Operations\_dataset')

def create\_output\_dir(output\_folder\_name):

    output\_folder = './' + output\_folder\_name

    # Check if the directory exists

    if os.path.exists(output\_folder):

        # If it exists, remove it (and all its contents)

        shutil.rmtree(output\_folder)

        print(f"Directory {output\_folder} has been removed")

    else:

        # If the directory does not exist, do nothing

        print(f"Directory {output\_folder} does not exist")

create\_output\_dir('output')

splitfolders.ratio('Automating\_Port\_Operations\_dataset', output='output', seed=1, \

                   ratio=(.8,.0,.2), group\_prefix=None)

list\_image\_files('output/train')

# list\_image\_files('output/val')

list\_image\_files('output/test')

train\_datagen = ImageDataGenerator(

    rescale=1./255,         # Normalize pixel values to [0, 1]

    validation\_split=0.2    # Reserve 20% of data for validation

)

# Directory containing the training images

train\_dir = 'output/train'

# Training generator

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

    train\_dir,

    target\_size=(224, 224),\

    batch\_size=32,\

    classes=['ferry\_boat', 'gondola', 'sailboat', 'cruise\_ship', 'kayak', \

              'inflatable\_boat', 'paper\_boat', 'buoy', 'freight\_boat'], \

    class\_mode='categorical',\

    shuffle=True,        # Use 'categorical' for multi-class classification

    subset='training'             # Set as training data

)

# Validation generator

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

    train\_dir,

   target\_size=(224, 224),\

    batch\_size=32,\

    classes=['ferry\_boat', 'gondola', 'sailboat', 'cruise\_ship', 'kayak', \

              'inflatable\_boat', 'paper\_boat', 'buoy', 'freight\_boat'], \

    class\_mode='categorical',\

    shuffle=True,        # Use 'categorical' for multi-class classification

    subset='validation'           # Set as validation data

)

# Create a separate ImageDataGenerator for test data with rescaling

test\_datagen = ImageDataGenerator(rescale=1./255)

# Directory containing the test images

test\_dir = 'output/test'

# Test generator

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

    test\_dir,

    target\_size=(224, 224),\

    batch\_size=32,\

    classes=['ferry\_boat', 'gondola', 'sailboat', 'cruise\_ship', 'kayak', \

              'inflatable\_boat', 'paper\_boat', 'buoy', 'freight\_boat'], \

    class\_mode='categorical',\

    shuffle=True,        # Use 'categorical' for multi-class classification

)

model = Sequential()

#adding first layer

model.add(Conv2D(32, (3,3), activation='relu', input\_shape=(224,224,3)))

model.add(MaxPooling2D(pool\_size=(2,2)))

#adding 2 layers

model.add(Conv2D(32, (3,3), activation='relu', input\_shape=(224,224,3)))

model.add(MaxPooling2D(pool\_size=(2,2)))

model.add(GlobalAveragePooling2D())

model.add(Dense(128, activation='relu'))

model.add(Dense(128, activation='relu'))

model.add(Dense(9, activation='softmax'))

model.summary()

model.compile(optimizer=Adam(learning\_rate=0.0001), \

                loss='categorical\_crossentropy', \

                metrics=['accuracy',Precision(),Recall()])

print(len(train\_generator), len(test\_generator))

print(test\_generator.classes)

model.fit(train\_generator, epochs=20)

# Evaluate the model on the test data

test\_loss, test\_accuracy,test\_precision,test\_recall  = model.evaluate(test\_generator, batch\_size=32)

# Print the test loss and accuracy

print(f'Test Loss: {test\_loss:.4f}')

print(f'Test Accuracy: {test\_accuracy:.4f}')

y\_pred\_prob = model.predict(test\_generator)

# Predicted classes

y\_pred = np.argmax(y\_pred\_prob, axis=1)

print(y\_pred)

f, ax = plt.subplots(figsize=(6, 4))

ConfusionMatrixDisplay.from\_predictions(test\_generator.classes, \

                                        y\_pred, \

                                        ax = ax, \

                                        normalize='true', \

                                        values\_format='.0%')

plt.title(f'Confusion matrix')

plt.show()

# print(confusion\_matrix(test\_generator.classes, y\_pred))

print(classification\_report(test\_generator.classes, y\_pred))

#Using pretrained model mobilenetv2 and do the same steps again

splitfolders.ratio('Automating\_Port\_Operations\_dataset', output='output', seed=1, \

                   ratio=(.7,.0,.3), group\_prefix=None)

list\_image\_files('output/train')

list\_image\_files('output/test')

train\_datagen1= ImageDataGenerator(rescale=1/255.0, validation\_split=0.2)

train\_generator1 = train\_datagen1.flow\_from\_directory(

  train\_dir,

  target\_size=(224,224),

  batch\_size=32,

  classes = ['ferry\_boat', 'gondola', 'sailboat', 'cruise\_ship', 'kayak', \

              'inflatable\_boat', 'paper\_boat', 'buoy', 'freight\_boat'], \

  class\_mode ='categorical',

  shuffle=True,

  subset ='training'

)

#Validation generator

validation\_generator1 = train\_datagen1.flow\_from\_directory(

  train\_dir,

  target\_size=(224,224),

  batch\_size=32,

  classes=['ferry\_boat', 'gondola', 'sailboat', 'cruise\_ship', 'kayak', \

              'inflatable\_boat', 'paper\_boat', 'buoy', 'freight\_boat'], \

  class\_mode= 'categorical',

  shuffle = True,

  subset ='validation'

)

test\_data\_gen1 = ImageDataGenerator(rescale=1/255.0)

test\_generator1 = test\_data\_gen1.flow\_from\_directory(

  test\_dir,

  target\_size=(224,224),

  batch\_size=32,

  classes=['ferry\_boat', 'gondola', 'sailboat', 'cruise\_ship', 'kayak', \

              'inflatable\_boat', 'paper\_boat', 'buoy', 'freight\_boat'], \

  class\_mode='categorical',

  shuffle=True

)

model = Sequential()

base\_model = MobileNet(weights='imagenet', include\_top=False)

average\_pooling\_layer = GlobalAveragePooling2D()(base\_model.output)

dropout\_layer\_1 = Dropout(rate=0.2)(average\_pooling\_layer)

hidden\_layer\_1 = Dense(256, activation='relu')(dropout\_layer\_1)

normalization\_layer\_1 = BatchNormalization()(hidden\_layer\_1)

dropout\_layer\_2 = Dropout(rate=0.1)(normalization\_layer\_1)

hidden\_layer\_2 = Dense(128, activation='relu')(dropout\_layer\_2)

normalization\_layer\_2 = BatchNormalization()(hidden\_layer\_2)

dropout\_layer\_3 = Dropout(rate=0.1)(normalization\_layer\_2)

predictions = Dense(9, activation='softmax')(dropout\_layer\_3)

model = Model(inputs=base\_model.input, outputs=predictions)

model.summary()

model.compile(optimizer=Adam(learning\_rate=0.0001), \

              loss='categorical\_crossentropy', \

              metrics=['accuracy',Precision(),Recall()])

early\_stopping\_cb = EarlyStopping(patience=5, restore\_best\_weights=True)

history = model.fit(train\_generator1,callbacks=[early\_stopping\_cb], epochs = 50)

history\_df = pd.DataFrame(history.history)

history\_df.plot(

    figsize=(8, 5), xlim=[0, early\_stopping\_cb.stopped\_epoch], ylim=[0, 1], \

    grid=True, xlabel="Epoch",

    style=["r--", "r--.", "b-", "b-\*"])

plt.legend(loc="lower right")

plt.title('Training/Validation loss & accuracy measured over each epoch')

plt.show()

test\_loss, test\_acc, test\_precision, test\_recall = \

    model.evaluate(test\_generator1, steps=test\_generator1.samples // 32)

print('Metrics obtained on test images')

print(f'Accuracy: {test\_acc}, Loss: {test\_loss}, ' +\

      f'Precision: {test\_precision}, Recall: {test\_recall}')

y\_test\_proba = model.predict(test\_generator1)

y\_test\_pred = np.argmax(y\_test\_proba, axis=1)

y\_test = test\_generator1.classes

print(classification\_report(y\_test, y\_test\_pred))

f, ax = plt.subplots(figsize=(6, 4))

ConfusionMatrixDisplay.from\_predictions(y\_test, \

                                        y\_test\_pred, \

                                        ax = ax, \

                                        normalize='true', \

                                        values\_format='.0%')

plt.title(f'Confusion matrix')

plt.show()

# We created two models.

# A full custom model

# A model baased on MobileNet with top layers replaced

# </ol> The accuracy obtained with a full custom model is much lower than that obtained with MobileNet pre-trained model. For automating port operations, the MobileNet custom model can be used and produces much better results.