from sklearn.datasets import load\_files

from keras.utils import np\_utils

import numpy as np

from glob import glob

tar=3

path='./dataset/'

# define function to load train, test, and validation datasets

def load\_dataset(path):

data = load\_files(path)

files = np.array(data['filenames'])

targets = np\_utils.to\_categorical(np.array(data['target']), tar)

return files, targets

# load train, test, and validation datasets

train\_files, train\_targets = load\_dataset(path)

test\_files=train\_files

test\_targets = train\_targets

# get the burn classes

# We only take the characters from a starting position to remove the path

#burn\_classes = [item[11:-1] for item in sorted(glob(path))]

burn\_classes = [item[15:-1] for item in sorted(glob("./dataset/\*/"))]

# print statistics about the dataset

print('There are %d total categories.' % len(burn\_classes))

print(burn\_classes)

print('There are %s total burn images.\n' % len(np.hstack([train\_files, test\_files])))

print('There are %d training images.' % len(train\_files))

print('There are %d test images.'% len(test\_files))

for file in train\_files: assert('.DS\_Store' not in file)

from tensorflow.keras.preprocessing import image

from tqdm import tqdm

# Note: modified these two functions, so that we can later also read the inception tensors which

# have a different format

def path\_to\_tensor(img\_path, width=224, height=224):

# loads RGB image as PIL.Image.Image type

img = image.load\_img(img\_path, target\_size=(width, height))

# convert PIL.Image.Image type to 3D tensor with shape (width, heigth, 3)

x = image.img\_to\_array(img)

# convert 3D tensor to 4D tensor with shape (1, width, height, 3) and return 4D tensor

return np.expand\_dims(x, axis=0)

def paths\_to\_tensor(img\_paths, width=224, height=224):

list\_of\_tensors = [path\_to\_tensor(img\_path, width, height) for img\_path in tqdm(img\_paths)]

return np.vstack(list\_of\_tensors)

import keras

import timeit

# graph the history of model.fit

def show\_history\_graph(history):

# summarize history for accuracy

plt.plot(history.history['accuracy'])

plt.plot(history.history['val\_accuracy'])

plt.title('model accuracy')

plt.ylabel('accuracy')

plt.xlabel('epoch')

plt.legend(['train', 'validation'], loc='upper left')

plt.show()

# summarize history for loss

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.title('model loss')

plt.ylabel('loss')

plt.xlabel('epoch')

plt.legend(['train', 'validation'], loc='upper left')

plt.show()

# callback to show the total time taken during training and for each epoch

class EpochTimer(keras.callbacks.Callback):

train\_start = 0

train\_end = 0

epoch\_start = 0

epoch\_end = 0

def get\_time(self):

return timeit.default\_timer()

def on\_train\_begin(self, logs={}):

self.train\_start = self.get\_time()

def on\_train\_end(self, logs={}):

self.train\_end = self.get\_time()

print('Training took {} seconds'.format(self.train\_end - self.train\_start))

def on\_epoch\_begin(self, epoch, logs={}):

self.epoch\_start = self.get\_time()

def on\_epoch\_end(self, epoch, logs={}):

self.epoch\_end = self.get\_time()

print('Epoch {} took {} seconds'.format(epoch, self.epoch\_end - self.epoch\_start))

from PIL import ImageFile

ImageFile.LOAD\_TRUNCATED\_IMAGES = True

# pre-process the data for Keras

train\_tensors = paths\_to\_tensor(train\_files).astype('float32')/255

test\_tensors = paths\_to\_tensor(test\_files).astype('float32')/255

from tensorflow.keras.layers import Conv2D, MaxPooling2D, GlobalAveragePooling2D

from tensorflow.keras.layers import Dropout, Flatten, Dense

from tensorflow.keras.models import Sequential

from tensorflow.keras.models import Model

from tensorflow.keras.callbacks import ModelCheckpoint

import matplotlib.pyplot as plt

img\_width, img\_height = 224, 224

batch\_size = 8

epoch=50

########

img\_width, img\_height = img\_width, img\_height

batch\_size = 32

samples\_per\_epoch = 10

validation\_steps = 300

nb\_filters1 = 32

nb\_filters2 = 64

conv1\_size = 3

conv2\_size = 3

pool\_size = 3

lr = 0.0004

from tensorflow.keras import optimizers

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dropout, Flatten, Dense, Activation

from tensorflow.keras.layers import Convolution2D, MaxPooling2D

from tensorflow.keras import callbacks

import time

#input\_shape=(img\_width, img\_height,3)

model = Sequential()

model.add(Convolution2D(nb\_filters1, conv1\_size, conv1\_size, padding='same', input\_shape=(img\_width, img\_height, 3)))

model.add(Activation("relu"))

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

model.add(Convolution2D(nb\_filters2, conv2\_size, conv2\_size, padding='same'))

model.add(Activation("relu"))

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

model.add(Flatten())

model.add(Dense(256))

model.add(Activation("relu"))

model.add(Dropout(0.5))

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

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

optimizer=optimizers.RMSprop(lr=lr),

metrics=['accuracy'])

hist=model.fit(train\_tensors, train\_targets ,validation\_split=0.1, epochs=epoch, batch\_size=64)

show\_history\_graph(hist)

test\_loss, test\_acc = model.evaluate(train\_tensors, train\_targets)

y\_pred=model.predict(train\_tensors)

from sklearn.metrics import confusion\_matrix,accuracy\_score

cm = confusion\_matrix(np.argmax(train\_targets, axis=1),np.argmax(y\_pred, axis=1))

from sklearn.metrics import roc\_curve

# Calculate ROC curve from y\_test and pred

fpr, tpr, thresholds = roc\_curve(np.argmax(test\_targets, axis=1)>=1,np.argmax(y\_pred, axis=1)>=1)

accuracycnn = accuracy\_score(np.argmax(test\_targets, axis=1),np.argmax(y\_pred, axis=1))

print("CNN confusion matrics=",cm)

print(" ")

print("CNN accuracy=",accuracycnn\*100)

# Plot the ROC curve

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

plt.title('Receiver Operating Characteristic')

# Plot ROC curve

plt.plot(fpr, tpr, label='l1')

plt.legend(loc='lower right')

# Diagonal 45 degree line

plt.plot([0,1],[0,1],'k--')

# Axes limits and labels

plt.xlim([-0.1,1.1])

plt.ylim([-0.1,1.1])

plt.ylabel('CNN True Positive Rate')

plt.xlabel('CNN False Positive Rate')

plt.show()

#model.save('color\_trained\_modelDNN.h5')

model.save('trained\_model\_CNN.h5')

#############