PROJECT CODE

#reading a csv file and extracting labels

```
import tensorflow as tf
import csv
fields=[]
rows=[]
with open('C:\\Users\\devishriarugula\\Desktop\\Major
Project\\TrainLabels.csv') as csvfile:
    read = csv.reader(csvfile)
    fields=next(read)
    for row in read:
        rows.append(row)
print(read.line_num)
print(fields)
attentive=[]
clipid=[]

for i in rows[:]:
    clipid.append(i[0])
    c=1
    attention=""
    for j in i[1:]:
        if int(j)>0:
            if len(attention)>0:
                attention=attention+"-"+fields[c]
            else:
                attention=fields[c]
        c=c+1
    attentive.append(attention)
print(attentive)
print(len(clipid))

#creating a dictionary for video and labels
dict={}
for i in range(5358):
    dict[clipid[i]]=attentive[i]
print(dict)

#frames.py
extracting the videos converting them to images and assigning labels

import cv2
import glob
import os
import pandas as pd
image=[]
labels=[]
files=glob.glob("C:\\Users\\devishriarugula\\Desktop\\Major
Project\\200050/*/*",recursive=True)
count=0
for myfile in files:
    fname=os.path.split(myfile)
    for i in dict.keys():
        if(i==fname[1]):
            frame1=dict[i]
    cam=cv2.VideoCapture(myfile)
```

```
    try:
        if not os.path.exists('stud1'):
            os.makedirs('stud1')
    except OSError:
        print("e")
    cf=0
    while(True):
        cam.set(cv2.CAP_PROP_POS_MSEC,(cf*1000))
        ret,frame=cam.read()
        if ret:
            name='./stud1/'+'frame'+str(count)+frame1+'.jpg'
            image.append(name)
            labels.append(frame1)
            print('creating'+name)
            cv2.imwrite(name, frame)
            cf=cf+1
            count=count+1
        else:
            break
cam.release()
cv2.destroyAllWindows()

data = pd.DataFrame()
data['image']=image
data['labels']=labels
data.to_csv('labels.csv',header=True, index=False)

#new python file to read labels
from imutils import paths
import matplotlib.pyplot as plt
import numpy as np
import pickle
import cv2
import os
import pandas as pd

print("[INFO] loading images...")
imagePaths = list(paths.list_images("C:\\Users\\devishriarugula\\stud1"))
data = []
labels = []
# loop over the image paths
for imagePath in imagePaths:
    image = cv2.imread(imagePath)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    image = cv2.resize(image, (224, 224))
    data.append(image)


from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
import pandas as pd
data = np.array(data)
print(data.shape)
train = pd.read_csv('labels.csv')

y=train['labels']
labels=np.array(y)
```

```
print(labels.shape)
lb = LabelBinarizer()
labels = lb.fit_transform(labels)
(trainX, testX, trainY, testY) = train_test_split(data,
labels,test_size=0.25)
print(y)

from keras.preprocessing.image import ImageDataGenerator
from keras.layers.pooling import AveragePooling2D
from keras.applications import ResNet50
from keras.layers.core import Dropout
from keras.layers.core import Flatten
from keras.layers.core import Dense
from keras.layers import Input
from keras.models import Model
from keras.optimizers import SGD
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

# initialize the training data augmentation object
trainAug = ImageDataGenerator(
      rotation_range=30,
      zoom_range=0.15,
      width_shift_range=0.2,
      height_shift_range=0.2,
      shear_range=0.15,
      horizontal_flip=True,
      fill_mode="nearest")
# initialize the validation/testing data augmentation object (which
# we'll be adding mean subtraction to)
valAug = ImageDataGenerator()
# define the ImageNet mean subtraction (in RGB order) and set the
# the mean subtraction value for each of the data augmentation
# objects
mean = np.array([123.68, 116.779, 103.939], dtype="float32")
trainAug.mean = mean
valAug.mean = mean

# load the ResNet-50 network, ensuring the head FC layer sets are left
# off
baseModel = ResNet50(weights="imagenet",
include_top=False,input_tensor=Input(shape=(224, 224, 3)))
# construct the head of the model that will be placed on top of the
# the base model
headModel = baseModel.output
headModel = AveragePooling2D(pool_size=(7, 7))(headModel)
headModel = Flatten(name="flatten")(headModel)
headModel = Dense(512, activation="relu")(headModel)
headModel = Dropout(0.5)(headModel)
headModel = Dense(len(lb.classes_), activation="softmax")(headModel)
# place the head FC model on top of the base model (this will become
# the actual model we will train)
model = Model(inputs=baseModel.input, outputs=headModel)
# loop over all layers in the base model and freeze them so they will
# *not* be updated during the training process
for layer in baseModel.layers:
      layer.trainable = False
```

```
import argparse
ap = argparse.ArgumentParser()

ap.add_argument("-e", "--epochs", type=int, default=25,help="# of epochs
to train our network for")

args = vars(ap.parse_args())

# compile our model (this needs to be done after our setting our
# layers to being non-trainable)
print("[INFO] compiling model...")
opt = SGD(lr=1e-4, momentum=0.9, decay=1e-4 / args["epochs"])
model.compile(loss="categorical_crossentropy",
optimizer=opt,metrics=["accuracy"])

print("[INFO] training head...")
H = model.fit_generator(trainAug.flow(trainX, trainY,
batch_size=32),steps_per_epoch=len(trainX) //
32,validation_data=valAug.flow(testX, testY),validation_steps=len(testX)
// 32,epochs=args["epochs"])

print("[INFO] evaluating network...")
predictions = model.predict(testX, batch_size=32)
print(classification_report(testY.argmax(axis=1),predictions.argmax(axis=
1), target_names=lb.classes_))

# serialize the model to disk


print("[INFO] serializing network...")
model.save('Model1')
# serialize the label binarizer to disk
f = open('manisha1', "wb")
f.write(pickle.dumps(lb))
f.close()

from keras.models import load_model
from collections import deque
import numpy as np

ap.add_argument("-s", "--size", type=int, default=128,help="size of queue
for averaging")
args = vars(ap.parse_args())
# load the trained model and label binarizer from disk
print("[INFO] loading model and label binarizer...")
model = load_model('Model1')
lb = pickle.loads(open('manisha1', "rb").read())
# initialize the image mean for mean subtraction along with the
# predictions queue
mean = np.array([123.68, 116.779, 103.939][::1], dtype="float32")
Q = deque(maxlen=args["size"])

# initialize the video stream, pointer to output video file, and
# frame dimensions
vs = cv2.VideoCapture('C:\\Users\\devishriarugula\\Desktop\\New
folder\\2000501020\\2000501020.avi')
writer = None
```

```
(W, H) = (None, None)
currentframe=0
while True:
    vs.set(cv2.CAP_PROP_POS_MSEC,(currentframe*1000))
    (grabbed, frame) = vs.read()
    print(grabbed)
    if not grabbed:
        break
    if W is None or H is None:
        (H, W) = frame.shape[:2]
    currentframe=currentframe+1
    output = frame.copy()
    frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    frame = cv2.resize(frame, (224, 224)).astype("float32")
    frame -= mean
    preds = model.predict(np.expand_dims(frame, axis=0))[0]
    Q.append(preds)
    results = np.array(Q).mean(axis=0)
    i = np.argmax(results)
    label = lb.classes_[i]

    text = "activity: {}".format(label)
    cv2.putText(output, text, (35, 50), cv2.FONT_HERSHEY_SIMPLEX,1.25,
(0, 255, 0), 5)
      # check if the video writer is None
    if writer is None:
        fourcc = cv2.VideoWriter_fourcc(*"mp4v")
        writer = cv2.VideoWriter("output1.", fourcc, 30,(W, H), True)
      # write the output frame to disk
    writer.write(output)
      # show the output image
    cv2.imshow("Output", output)
    key = cv2.waitKey(1) & 0xFF
      # if the `q` key was pressed, break from the loop
    if key == ord("q"):
        break
# release the file pointers
print("[INFO] cleaning up...")
writer.release()
vs.release()
```