**SOURCE CODE**

import argparse

import imutils

import pyttsx3

import pickle

import time

import cv2

import urllib.request

import urllib.parse

def sendSMS(apikey, numbers, sender, message):

data = urllib.parse.urlencode({'username':username,'apikey': apikey, 'numbers': numbers, 'message' : message, 'sender': sender})

data = data.encode('utf-8')

fr = f.read()

return(fr)

# construct the argument parser and parse the arguments

ap = argparse.ArgumentParser()

ap.add\_argument("-c", "--conf", required=True,

help="Path to the input configuration file")

args = vars(ap.parse\_args())

# load the configuration file

conf = Conf(args["conf"])

# initialize consecutive recognition count to 0

consecCount = 0

student1 = akshay"

student2 = "saiteja"

#student3 = "3"

unknown\_flag=0

student1\_flag=0

student2\_flag=0

#student3\_flag=0

process\_flag = 0

def create\_msg():

msg = MIMEMultipart()

msg['Subject'] = 'Class1'

#msg['From'] = 'akshay@gmail.com'

#msg['To'] = '@mail.cc'

text = MIMEText("Class1")

msg.attach(text)

return msg

def create\_msg2():

msg1 = MIMEMultipart()

msg1['Subject'] = 'Class2'

#msg['From'] = 'akshay@gmail.com'

#msg['To'] = '@mail.cc'

text = MIMEText("Class2")

msg1.attach(text)

return msg1

def attach\_file(msg\_cont):

f = open(file\_path)

image = MIMEText(f.read())

msg\_cont.attach(image)

return msg\_cont

def attach\_file2(msg\_cont):

f = open(file\_path2)

image = MIMEText(f.read())

msg\_cont.attach(image)

return msg\_cont

def SendMail(msg):

s = smtplib.SMTP(Server)

s.ehlo()

s.starttls()

s.ehlo()

s.login(UserName, UserPassword)

s.sendmail("akshay@gmail.com", "saiteja@gmail.com", msg.as\_string())

s.quit()

# initialize the database, student table, and attendance table

# objects

db = TinyDB(conf["db\_path"])

studentTable = db.table("student")

attendanceTable = db.table("attendance")

# load the actual face recognition model along with the label encoder

recognizer = pickle.loads(open(conf["recognizer\_path"], "rb").read())

le = pickle.loads(open(conf["le\_path"], "rb").read())

# initialize the video stream and allow the camera sensor to warmup

print("[INFO] warming up camera...")

vs = VideoStream(src=0).start()

#vs = VideoStream(usePiCamera=True).start()

time.sleep(2.0)

# initialize previous and current person to None

file\_path = "a.txt"

file\_path2 = "b.txt"

prevPerson = None

curPerson = None

# initialize consecutive recognition count to 0

#consecCount = 0

# initialize the text-to-speech engine, set the speech language, and

# the speech rate

# initialize a dictionary to store the student ID and the time at

# which their attendance was taken

#studentDict = {}

# loop over the frames from the video stream

while True:

if process\_flag==0:

# store the current time and calculate the time difference

# between the current time and the time for the class

currentTime = datetime.now()

timeDiff = (currentTime - datetime.strptime(conf["timing"],

"%H:%M")).seconds

# grab the next frame from the stream, resize it and flip it

# horizontally

frame = vs.read()

frame = imutils.resize(frame, width=400)

frame = cv2.flip(frame, 1)

# if the maximum time limit to record attendance has been crossed

# then skip the attendance taking procedure

if timeDiff > conf["max\_time\_limit"]:

# check if the student dictionary is not empty

#if len(studentDict) != 0:

# insert the attendance into the database and reset the

# student dictionary

#attendanceTable.insert({str(date.today()): studentDict})

#studentDict = {}

# draw info such as class, class timing, and current time on

# the frame

cv2.putText(frame, "Class: {}".format(conf["class"]),

(10, 10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 0, 255), 1)

cv2.putText(frame, "Class timing: {}".format(conf["timing"]),

(10, 25), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 0, 255), 1)

cv2.putText(frame, "Current time: {}".format(

currentTime.strftime("%H:%M:%S")), (10, 40),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 0, 255), 1)

# show the frame

cv2.imshow("Attendance System", frame)

key = cv2.waitKey(1) & 0xFF

# if the `q` key was pressed, break from the loop

if key == ord("q"):

break

# skip the remaining steps since the time to take the

# attendance has ended

continue

# convert the frame from RGB (OpenCV ordering) to dlib

# ordering (RGB)

rgb = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

# detect the (x, y)-coordinates of the bounding boxes

# corresponding to each face in the input image

boxes = face\_recognition.face\_locations(rgb,

model=conf["detection\_method"])

# loop over the face detections

for (top, right, bottom, left) in boxes:

# draw the face detections on the frame

cv2.rectangle(frame, (left, top), (right, bottom),

(0, 255, 0), 2)

# calculate the time remaining for attendance to be taken

timeRemaining = conf["max\_time\_limit"] - timeDiff

# draw info such as class, class timing, current time, and

# remaining attendance time on the frame

cv2.putText(frame, "Class: {}".format(conf["class"]), (10, 10),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 0, 255), 1)

cv2.putText(frame, "Class timing: {}".format(conf["timing"]),

(10, 25), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 0, 255), 1)

cv2.putText(frame, "Current time: {}".format(

currentTime.strftime("%H:%M:%S")), (10, 40),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 0, 255), 1)

cv2.putText(frame, "Time remaining: {}s".format(timeRemaining),

(10, 55), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 0, 255), 1)

# check if atleast one face has been detected

if len(boxes) > 0:

# compute the facial embedding for the face

encodings = face\_recognition.face\_encodings(rgb, boxes)

preds = recognizer.predict\_proba(encodings)[0]

j = np.argmax(preds)

curPerson = le.classes\_[j]

# if the person recognized is the same as in the previous

# frame then increment the consecutive count

if prevPerson == curPerson:

consecCount += 1

# otherwise, these are two different people so reset the

# consecutive count

else:

consecCount = 0

# set current person to previous person for the next

# iteration

prevPerson = curPerson

# if a particular person is recognized for a given

# number of consecutive frames, we have reached a

# positive recognition and alert/greet the person accordingly

if consecCount >= conf["consec\_count"]:

# check if the student's attendance has been already

# taken, if not, record the student's attendance

#if curPerson not in studentDict.keys():

#studentDict[curPerson] = datetime.now().strftime("%H:%M:%S")

# get the student's name from the database and let them

# know that their attendance has been taken

name = studentTable.search(where(

curPerson))[0][curPerson][0]

# construct a label saying the student has their attendance

# taken and draw it on to the frame

label = "{}, you are now marked as present in {}".format(

name, conf["class"])

cv2.putText(frame, label, (5, 175),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (255, 0, 0), 2)

if name == student1:

student1\_flag=1

consecCount = 0

elif name == student2:

student2\_flag=1

consecCount = 0

elif name == 'unknown':

if unknown\_flag == 0:

unknown\_flag=1

if unknown\_flag ==1:

#resp = sendSMS(apikey, numbers,sender, message)

#print(resp)

unknown\_flag=2

else:

# construct a label asking the student to stand in fron

# to the camera and draw it on to the frame

label = "Please stand in front of the camera"

cv2.putText(frame, label, (5, 175),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (255, 0, 0), 2)

# show the frame

cv2.imshow("Survilance System", frame)

key = cv2.waitKey(1) & 0xFF

# check if the `q` key was pressed

if key == ord("q"):

# check if the student dictionary is not empty, and if so,

# insert the attendance into the database

if len(studentDict) != 0:

attendanceTable.insert({str(date.today()): studentDict})

# break from the loop

break

if timeDiff2 == conf["max\_time\_limit"]:

f = open("log.txt", "a+")

if student1\_flag == 1:

f.write("\n" + student1 +" "+ "is present")

else:

f.write("\n"+ student1 +" "+"is absent")

if student2\_flag == 1:

f.write("\n"+ student2 +" "+ "is present")

else:

f.write("\n"+ student2 +" "+"is absent")

f.close()

msg\_head = create\_msg()

attach=attach\_file(msg\_head)

#SendMail(attach)

process\_flag=1

student1\_flag=0

student2\_flag=0

elif process\_flag==1:

# store the current time and calculate the time difference

# between the current time and the time for the class

currentTime = datetime.now()

timeDiff2 = (currentTime - datetime.strptime(conf["timing2"],

"%H:%M")).seconds

# grab the next frame from the stream, resize it and flip it

# horizontally

frame = vs.read()

frame = imutils.resize(frame, width=400)

frame = cv2.flip(frame, 1)

# if the maximum time limit to record attendance has been crossed

# then skip the attendance taking procedure

if timeDiff2 > conf["max\_time\_limit2"]:

# check if the student dictionary is not empty

#if len(studentDict) != 0:

# insert the attendance into the database and reset the

# student dictionary

#attendanceTable.insert({str(date.today()): studentDict})

#studentDict = {}

# draw info such as class, class timing, and current time on

# the frame

cv2.putText(frame, "Class: {}".format(conf["class2"]),

(10, 10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 0, 255), 1)

cv2.putText(frame, "Class timing: {}".format(conf["timing2"]),

(10, 25), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 0, 255), 1)

cv2.putText(frame, "Current time: {}".format(

currentTime.strftime("%H:%M:%S")), (10, 40),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 0, 255), 1)

# show the frame

cv2.imshow("Attendance System", frame)

key = cv2.waitKey(1) & 0xFF

# if the `q` key was pressed, break from the loop

if key == ord("q"):

break

# skip the remaining steps since the time to take the

# attendance has ended

continue

# convert the frame from RGB (OpenCV ordering) to dlib

# ordering (RGB)

rgb = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

# detect the (x, y)-coordinates of the bounding boxes

# corresponding to each face in the input image

boxes = face\_recognition.face\_locations(rgb,

model=conf["detection\_method"])

# loop over the face detections

for (top, right, bottom, left) in boxes:

# draw the face detections on the frame

cv2.rectangle(frame, (left, top), (right, bottom),

(0, 255, 0), 2)

# calculate the time remaining for attendance to be taken

timeRemaining = conf["max\_time\_limit2"] - timeDiff2

# draw info such as class, class timing, current time, and

# remaining attendance time on the frame

cv2.putText(frame, "Class: {}".format(conf["class2"]), (10, 10),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 0, 255), 1)

cv2.putText(frame, "Class timing: {}".format(conf["timing2"]),

(10, 25), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 0, 255), 1)

cv2.putText(frame, "Current time: {}".format(

currentTime.strftime("%H:%M:%S")), (10, 40),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 0, 255), 1)

cv2.putText(frame, "Time remaining: {}s".format(timeRemaining),

(10, 55), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 0, 255), 1)

# check if atleast one face has been detected

if len(boxes) > 0:

# compute the facial embedding for the face

encodings = face\_recognition.face\_encodings(rgb, boxes)

preds = recognizer.predict\_proba(encodings)[0]

j = np.argmax(preds)

curPerson = le.classes\_[j]

# if the person recognized is the same as in the previous

# frame then increment the consecutive count

if prevPerson == curPerson:

consecCount += 1

# otherwise, these are two different people so reset the

# consecutive count

else:

consecCount = 0

# set current person to previous person for the next

# iteration

prevPerson = curPerson

# if a particular person is recognized for a given

# number of consecutive frames, we have reached a

# positive recognition and alert/greet the person accordingly

if consecCount >= conf["consec\_count"]:

# check if the student's attendance has been already

# taken, if not, record the student's attendance

#if curPerson not in studentDict.keys():

#studentDict[curPerson] = datetime.now().strftime("%H:%M:%S")

# get the student's name from the database and let them

# know that their attendance has been taken

name = studentTable.search(where(

curPerson))[0][curPerson][0]

# construct a label saying the student has their attendance

# taken and draw it on to the frame

label = "{}, you are now marked as present in {}".format(

name, conf["class2"])

cv2.putText(frame, label, (5, 175),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (255, 0, 0), 2)

if name == student1:

student1\_flag=1

consecCount = 0

elif name == student2:

student2\_flag=1

consecCount = 0

elif name == 'unknown':

if unknown\_flag == 0:

unknown\_flag=1

if unknown\_flag ==1:

#resp = sendSMS(apikey, numbers,sender, message)

#print(resp)

unknown\_flag=2

else:

# construct a label asking the student to stand in fron

# to the camera and draw it on to the frame

label = "Please stand in front of the camera"

cv2.putText(frame, label, (5, 175),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (255, 0, 0), 2)

# show the frame

cv2.imshow("Survilance System", frame)

key = cv2.waitKey(1) & 0xFF

# check if the `q` key was pressed

if key == ord("q"):

# check if the student dictionary is not empty, and if so,

# insert the attendance into the database

if len(studentDict) != 0:

attendanceTable.insert({str(date.today()): studentDict})

# break from the loop

break

if timeDiff2 == conf["max\_time\_limit2"]:

f = open("a.txt", "a+")

if student1\_flag == 1:

f.write("\n" + student1 +" "+ "is present")

else:

f.write("\n"+ student1 +" "+"is absent")

if student2\_flag == 1:

f.write("\n"+ student2 +" "+ "is present")

else:

f.write("\n"+ student2 +" "+"is absent")

f.close()

msg\_head = create\_msg()

attach=attach\_file(msg\_head)

#SendMail(attach)

process\_flag=1

student1\_flag=0

student2\_flag=0

print("[INFO] cleaning up...")

print('label', label)

print("student1\_flag,student2\_flag,student3\_flag", student1\_flag,student2\_flag,student3\_flag)

# clean up

print("[INFO] cleaning up...")

vs.stop()

db.close()