import face\_recognition

import cv2

import numpy as np

import glob

import time

import csv

import pickle

# import mysql.connector

# import datetime

f=open("ref\_name.pkl","rb")

ref\_dictt=pickle.load(f) #ref\_dict=ref vs name

f.close()

f=open("ref\_embed.pkl","rb")

embed\_dictt=pickle.load(f) #embed\_dict- ref vs embedding

f.close()

############################################################################ encodings and ref\_ids

known\_face\_encodings = [] #encodingd of faces

known\_face\_names = [] #ref\_id of faces

for ref\_id , embed\_list in embed\_dictt.items():

for embed in embed\_list:

known\_face\_encodings +=[embed]

known\_face\_names += [ref\_id]

#############################################################frame capturing from camera and face recognition

video\_capture = cv2.VideoCapture(0)

# Initialize some variables

face\_locations = []

face\_encodings = []

face\_names = []

process\_this\_frame = True

while True :

# Grab a single frame of video

ret, frame = video\_capture.read()

# Resize frame of video to 1/4 size for faster face recognition processing

small\_frame = cv2.resize(frame, (0, 0), fx=0.25, fy=0.25)

# Convert the image from BGR color (which OpenCV uses) to RGB color (which face\_recognition uses)

rgb\_small\_frame = small\_frame[:, :, ::-1]

# Only process every other frame of video to save time

if process\_this\_frame:

# Find all the faces and face encodings in the current frame of video

face\_locations = face\_recognition.face\_locations(rgb\_small\_frame)

face\_encodings = face\_recognition.face\_encodings(rgb\_small\_frame, face\_locations)

face\_names = []

for face\_encoding in face\_encodings:

# See if the face is a match for the known face(s)

matches = face\_recognition.compare\_faces(known\_face\_encodings, face\_encoding)

name = "Unknown"

# # If a match was found in known\_face\_encodings, just use the first one.

# if True in matches:

# first\_match\_index = matches.index(True)

# name = known\_face\_names[first\_match\_index]

# Or instead, use the known face with the smallest distance to the new face

face\_distances = face\_recognition.face\_distance(known\_face\_encodings, face\_encoding)

best\_match\_index = np.argmin(face\_distances)

if matches[best\_match\_index]:

name = known\_face\_names[best\_match\_index]

face\_names.append(name)

process\_this\_frame = not process\_this\_frame

# Display the results

for (top, right, bottom, left), name in zip(face\_locations, face\_names):

# Scale back up face locations since the frame we detected in was scaled to 1/4 size

top \*= 4

right \*= 4

bottom \*= 4

left \*= 4

#updating in database

cv2.rectangle(frame, (left, top), (right, bottom), (0, 0, 255), 2)

# Draw a label with a name below the face

cv2.rectangle(frame, (left, bottom - 35), (right, bottom), (0, 0, 255), cv2.FILLED)

font = cv2.FONT\_HERSHEY\_DUPLEX

cv2.putText(frame, ref\_dictt[name], (left + 6, bottom - 6), font, 1.0, (255, 255, 255), 1)

font = cv2.FONT\_HERSHEY\_DUPLEX

# cv2.putText(frame, last\_rec[0], (6,20), font, 1.0, (0,0 ,0), 1)

# Display the resulting imagecv2.putText(frame, ref\_dictt[name], (left + 6, bottom - 6), font, 1.0, (255, 255, 255), 1)

cv2.imshow('Video', frame)

# Hit 'q' on the keyboard to quit!

if cv2.waitKey(1) & 0xFF == ord('q'):

# t.cancel()

break

# break

# Release handle to the webcam

video\_capture.release()

cv2.destroyAllWindows()