**A picture containing schematic

Description automatically generated**

**Department of Electrical Engineering and Computer Science**

**COSC 330 – Introduction to Artificial Intelligence**

**Mini Project – Face Recognition**

Group Members

Yaphet Elias 100049897

Nahom Tesfu 100049900

Tiemar Semere 100049901

# Abstract

# Problem Description

A system that can be deployed in in-doors premises for both identity control and watching for suspects is required to be developed for a security company. This system is a facial recognition system that employs machine learning and works with normal camera and Kinect camera.

# Introduction

Machine Learning is a field of study that gives computers the ability to learn without being explicitly programmed, Arthur Samuel (1959). In different industries, machine learning has paved the way for technological accomplishments ranging its applications from database mining, self-customizing programs, to computer vision. Face recognition is one application of machine learning where it works to identify people using their facial image. Basically, the system captures a 2D or a 3D image of the face, extracts from it a facial representation (a set of metrics representing the face), then matches this representation with one corresponding the claimed identity. *(rephrase… so disconn)*

Insert the diagram here…

# Solution – What it does and approach

*How does the application work…generally… (might exclude)*

First login page

Either login or signup

Sign up: to get the frame, call camera class,

Signing up means getting the images from the frame

These images will be used to train the model

Each time a person signs up, thread runs to train the model,

When signup is done(after training) – status sent from server, successful signup message on a status bar

Return to login, to login

Sign in – frame appears to get the image

The image(s) – a number of them to avoid misclassification and get the high prob

(assume there are already people)

*…*

*Generating the data – haar cascade… opencv*

As face recognition is a direct application of computer vision, OpenCV, an open source computer vision library that includes numerous computer vision algorithms, is used. OpenCV provides pretrained models, that can be read using a load method. First, a [cv2.CascadeClassifier](https://docs.opencv.org/3.4/d1/de5/classcv_1_1CascadeClassifier.html)() is created and the *haarcascade\_frontalface\_default.xml* file is loaded. Afterwards, the detection is done using the [cv2.CascadeClassifier.detectMultiScale](https://docs.opencv.org/3.4/d1/de5/classcv_1_1CascadeClassifier.html#aaf8181cb63968136476ec4204ffca498)() method, which returns boundary rectangles for the detected faces. Haar Cascade Classifiers are basically a machine learning based approaches where a cascade function is trained from a lot of images both positive and negative, and based on the training, are then used to detect the objects in the other images. For the case of this project, the *haarcascade\_frontalface\_default.xml* is the file that was used as it contains the features set to detect the frontal face. *(clean up)*

*Splitting the data to training and testing*

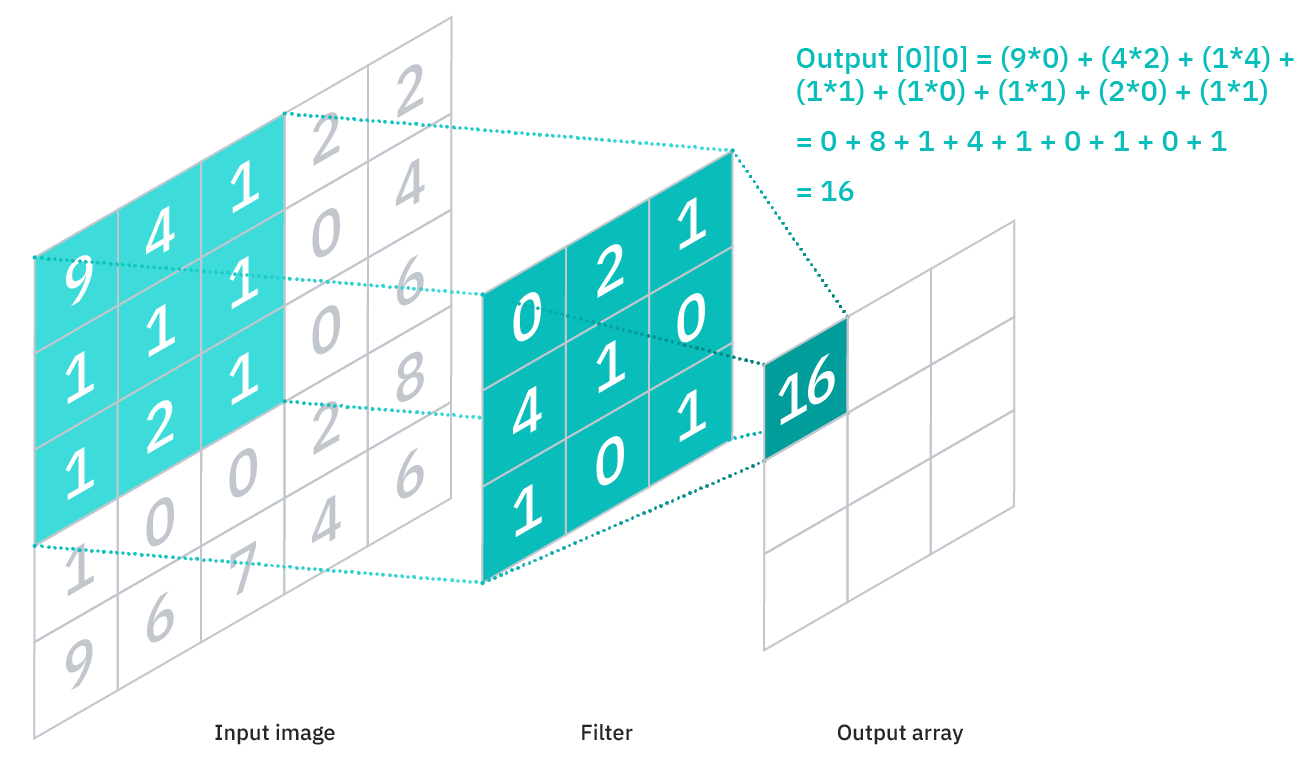
*…*

*Training the model = CNN*

Convolutional Neural Network is a form of neural network that offers better performance with image recognition.

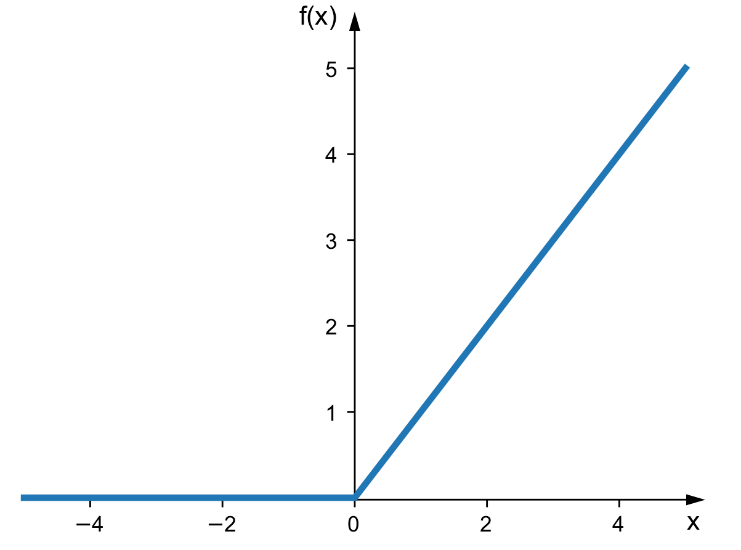
It has three main layers: convolutional, pooling and fully connected layer.

Convolutional layer is the layer where most of the computation takes place. It requires input data, a filter, and a feature map. [1] The image is made of 3D matrix of pixels corresponding to RGB image. The process known as convolution is when the filter (feature detector) swipes across the receptive fields or the matrix looking for a specific feature. The filter is a 2D array of weights representing part of the image. The dot product is calculated between the input pixels and the filter which is then fed into an output array. Afterwards, the filter shifts by a stride, repeating the process until the filter has swept across the entire image [1]. The output array is the feature map.



*Figure. A Convolutional layer with filter size 3x3*

The activation function used in this layer is the rectified linear activation function/unit (ReLU) because it allows to models to learn faster as it gives the effective ability to backpropagate the gradient information to the input layers of the model. ReLU outputs the input if the input is greater that 0, and outputs 0 if the input is less than or equal to 0.



*Figure. ReLU activation function*

Other activation functions(sigmoid and tanh) suffer from the vanishing gradient problem. That basically means as the number of nodes or layers in the neural network increase, the gradient of the loss function becomes smaller (vanishing) making the update on the weights small and this leads to slow learning of the model *(do I more be more elaborate about this??..I think this might be enough)..(which padding did we use.. The padding is zero (same)…what we used here… need the final code to write that)*

Pooling summarizes a set of adjacent units from the preceding layer with a single value by simply aggregating values within a respective field to populate the output array. It uses the same technique as the preceding layer, but the filter in this layer does not have any weights. Instead, the aggregation is carried out by making the filter either select the pixel with the maximum value to send to the output array, called max pooling or calculate the average value within the receptive field to send to the output array, called average pooling. The solution uses max pooling. (*… need the final code to confirm and add stuff*)….*Yeah it uses the max pooling (will include the code at the end of this report)*



Fully connected layer is a feed forward neural network. The input to the fully connected layer is the output from the final Pooling or Convolutional Layer, which is flattened (output from the final or any Pooling and Convolutional Layer is a 3-dimensional matrix, to flatten that is to unroll all its values into a vector) and then fed into it. The pixel values of the input image are not directly connected to the output layer in partially connected layers (pooling and convolutional layers). However, in the fully connected layer, each node in the output layer connects directly to a node in the previous layer. This layer performs the task of classification based on the features extracted through the previous layers and their different filters. While convolutional and pooling layers use ReLu functions, fully connected layers usually leverage a softmax activation function to get the probabilities of the input being a particular class. *(rephrase)*

Mulitclass classification problem requires a network with one node for each class in theop later and the sum of the predicted probabilities equals one.

Dropout; overfitting prevent

tsparse ce for *loss(what did we use??)*

*Sparse categorical cross entropy (I’ll have a look into some papers and will write something here)*

*what libraries did we use and why…?*

*Libraries.*

*For the model*

*Since our model is built in python, for the basic array operation we are using python libraries numpy and pandas, and for basic os related operation that include manipulating directories to store training data and trained model we are using python library os. Opencv us used to take and manage the pictures in our solution and we are creating and training our model using the keras and tensorflow models. To present the training data in a format that our model supports we are using methods from the sklearn libraries in python. As explained earlier once our solution takes the pictures of the user it saves them and trains the model in another thread which meant we have to use the threading library.*

*(Not sure how we are using tflearn but we should have an idea once its all done)*

import numpy

import cv2

import os

import pandas

from threading import Thread

import sklearn

import sklearn.model\_selection

import tensorflow as tf

import tflearn

from tflearn.layers.conv import conv\_2d, max\_pool\_2d

from tflearn.layers.core import input\_data, dropout, fully\_connected, flatten

from tflearn.layers.estimator import regression

from tensorflow import keras

from keras import layers

from cv2 import INTER\_AREA

*for the web app*

from flask import Flask, render\_template, request, Response, redirect, url\_for, jsonify

*For the web app, we are using the flask library.*

Prediction process

# Code(maybe don’t include here)

# Conclusion

Reference

[What are Convolutional Neural Networks? | IBM](https://www.ibm.com/cloud/learn/convolutional-neural-networks)

Artificial Intelligence, Modern Approach, Global Edition by Staurt J Russel, Peter Norvig

<https://analyticsindiamag.com/addressing-the-vanishing-gradient-problem-a-guide-for-beginners/#:~:text=The%20vanishing%20gradients%20problem%20is,input%20layers%20of%20the%20model>.

Camera:

import numpy

import cv2

import os

import pandas

from threading import Thread

import sklearn

import tensorflow as tf

import tflearn

from tflearn.layers.conv import conv\_2d, max\_pool\_2d

from tflearn.layers.core import input\_data, dropout, fully\_connected, flatten

from tflearn.layers.estimator import regression

from utilities import getmodel,getlabel,checkdir,getid,updateidfile,updatemodel,trainmodel,getuntrainedmodel,load\_data,getdir,getname

class Camera():

    def \_\_init\_\_(self,name):

        self.name = name

        self.ID = getid()

        self.picnum = 40 #number of pictures to take

        self.faceclassifier = cv2.CascadeClassifier("haarcascade\_frontalface\_default.xml")

        self.savedpics = 0

        self.model = getmodel()

        print("here")

        self.vid = cv2.VideoCapture(0)

        self.collected = []

    def destroy(self):

        self.vid.release()

    def getframe(self): #output (status (0 for not done and 1 for done), frame)

        checkdir()

        updateidfile(self.name,self.ID)

        directory = getdir(self.ID)

       # vid = cv2.VideoCapture(0)

        ret, frame = self.vid.read()

        gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

        faces = self.faceclassifier.detectMultiScale(gray, 1.3, 5)

        for x,y,w,h in faces:#should remove this since we are checking for only one face

            face = gray[y:y+h,x:x+w]

            if faces is not None:

                self.savedpics += 1

                face = cv2.resize(face, (200,200))

                file\_name\_path = os.path.join(directory,str(self.savedpics) + ".jpg")

                cv2.imwrite(file\_name\_path, face)

                cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)

        ret, retframe = cv2.imencode('.png', frame)#change this

        if(self.savedpics >= self.picnum):#done collecting the data

            #create a thread and run it here

            self.destroy()

            new\_thread = Thread(target=updatemodel)

            new\_thread.start()

            return (1,retframe.tobytes())

        else:

            return (0,retframe.tobytes())

    def getdir(ID):

        directory = 'Datadir'

        if (not os.path.isdir(directory)):

            os.mkdir(directory)

        directory = os.path.join(directory, str(ID))

        if (not os.path.isdir(directory)):

            os.mkdir(directory)

        return directory

    def detectface(self): #returns (status (1:found,0:searching,-1:not found,2:morethan two people, 3: no model), frame,id)

        ret, frame = self.vid.read()

        gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

        faces = self.faceclassifier.detectMultiScale(gray, 1.3, 5)

        self.model = getmodel()

        if(self.model is None):

            return (3,0,0,None)

        status =0

        if (len(faces)>1):

            status = 2

        for x,y,w,h in faces:#should remove this since we are checking for only one face

            face = gray[y:y+h,x:x+w]

            if faces is not None:

                face = cv2.resize(face, (50,50))

                #face = cv2.cvtColor(face, cv2.COLOR\_BGR2GRAY)

                face = numpy.reshape(face,(50,50,1))

                newfaces = []

                newfaces.append(face)

                newfaces = numpy.array(newfaces)

                print(newfaces.shape)

                result = self.model.predict(newfaces)

                print(result[0])

                print(numpy.argmax(result[0]))

                if(result[0][numpy.argmax(result[0])]<0.50): #could be changed to increase accuracy

                    label = 'Unknown'

                    self.collected.append(-1)

                else:

                    label = getlabel()[numpy.argmax(result)]

                    self.collected.append(numpy.argmax(result))

                cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)

        ret, retframe = cv2.imencode('.png', frame)

        if(len(self.collected)==10):

            self.destroy()

            freqcount = dict()

            for i in self.collected:

                if i in freqcount.keys():

                    freqcount[i] += 1

                else:

                    freqcount[i] = 1

            print(freqcount)

            k = max(freqcount, key = freqcount.get)

            print(k)

            if(freqcount[k] >= 5 and status!= 2):#succesfully found a match

                print(getname(getlabel()[k]))

                return (1,retframe.tobytes(),getlabel()[k],getname(getlabel()[k]))

            elif(status == 2):

                return (status,retframe.tobytes(),getlabel()[k],getname(getlabel()[k]))

            else:

                return (-1,retframe.tobytes(),0,None)

        return (0,retframe.tobytes(),0,None)

Utilities:

import numpy

import cv2

import os

import pandas

from threading import Thread

import sklearn

import sklearn.model\_selection

import tensorflow as tf

import tflearn

from tflearn.layers.conv import conv\_2d, max\_pool\_2d

from tflearn.layers.core import input\_data, dropout, fully\_connected, flatten

from tflearn.layers.estimator import regression

from tensorflow import keras

from keras import layers

from cv2 import INTER\_AREA

def getmodel():

    modeldir = os.path.join('Datadir','model.model')

    if(os.path.isfile(modeldir)):

        return tf.keras.models.load\_model(modeldir)

    else:

        return None

def getname(ID):

    directory = os.path.join('Datadir','mappingfile.csv')

    if(not os.path.isfile(directory)):

        print("test")

        return None

    mappingfile = pandas.read\_csv(directory)

    for i in range(len(mappingfile['ID'])):

        if(str(mappingfile['ID'][i]) == str(ID)):

            print(mappingfile['Name'][i])

            return mappingfile['Name'][i]

def getlabel():

    directory = 'Datadir'

    dirs=[x[1] for x in os.walk(directory)]

    return dirs[0]

def checkdir():

    directory = 'Datadir'

    if (not os.path.isdir(directory)):

        os.mkdir(directory)

def getid():

        directory = 'Datadir'

        file = os.path.join(directory,'mappingfile.csv')

        if( os.path.isfile(file) == False):

            ID = 13579

        else:

            name\_data = pandas.read\_csv(file)

            if(len(name\_data) == 0):

                ID = 3579

            else:

                print(name\_data)

                print(name\_data['ID'].iloc[-1])

                ID = name\_data['ID'].iloc[-1] + 1

        return ID

def updateidfile(name,ID):

        directory = 'Datadir'

        file = os.path.join(directory,'mappingfile.csv')

        values = {'Name': name, 'ID':ID}

        if( os.path.isfile(file) is False):

            filedata = pandas.DataFrame(values,index=[0])

            #filedata = numpy.array(filedata)

        else:

            filedata = pandas.read\_csv(file)

            print(filedata.values)

            if(ID in filedata.values):

                return

            filedata = filedata.append(values, ignore\_index = True)

            #addname

        filedata.to\_csv(file, index = False)

def updatemodel():

    #model = getuntrainedmodel(len(getlabel()))

    model = getanothermodel(len(getlabel()))

    trainmodel(model)

    directory = 'Datadir'

    filename = os.path.join(directory,'model.model')

    model.save(filename)

    #save model

def trainmodel(model):

    X\_train,X\_test, Y\_train,Y\_test = load\_data()

    X\_train,Y\_train,X\_test,Y\_test = numpy.array(X\_train), numpy.array(Y\_train), numpy.array(X\_test), numpy.array(Y\_test)

    print(X\_train.shape, Y\_train.shape)

    X\_train = X\_train.reshape(len(X\_train),50,50,1)

    X\_test = X\_test.reshape(len(X\_test),50,50,1)

    model.fit(X\_train, Y\_train,epochs=10)

    #preparing the data to use with tflearn

    #new\_y\_train = []

    #new\_y\_test = []

    #k = len(getlabel())

    #for i in Y\_train:

    #   encoded = numpy.zeros(k)

    #   encoded[i] = 1

    #   new\_y\_train.append(encoded)

    #for i in Y\_test:

    #   encoded = numpy.zeros(k)

    #   encoded[i] = 1

    #   new\_y\_test.append(encoded)

    #print(new\_y\_train)

    #model.fit(X\_train, new\_y\_train, n\_epoch=100, validation\_set=(X\_test, new\_y\_test), show\_metric = True)

def getanothermodel(num):

    model = tf.keras.models.Sequential()

    model.add(tf.keras.Input(shape=[50,50,1]))

    model.add(tf.keras.layers.Conv2D(64,3,activation="relu"))

    model.add(tf.keras.layers.MaxPooling2D(pool\_size=3))

    model.add(tf.keras.layers.Dense(32,activation="relu"))

    model.add(tf.keras.layers.Conv2D(128,3,activation="relu"))

    model.add(tf.keras.layers.MaxPooling2D(pool\_size=3))

    model.add(tf.keras.layers.Flatten())

    model.add(tf.keras.layers.Dense(1024,activation="relu"))

    model.add(tf.keras.layers.Dropout(0.5))

    model.add(tf.keras.layers.Dense(num,activation="softmax"))

    model.compile(

        optimizer="adam",

        loss="sparse\_categorical\_crossentropy",

        metrics=["accuracy"]

    )

    print(model.summary())

    return model

def getuntrainedmodel(num):

    convnet = input\_data(shape=[50,50,1])

    convnet = conv\_2d(convnet, 32, 3, activation='relu')

    convnet = max\_pool\_2d(convnet, 5)

    convnet = conv\_2d(convnet, 64, 3, activation='relu')

    convnet = max\_pool\_2d(convnet, 5)

    convnet = conv\_2d(convnet, 128, 3, activation='relu')

    convnet = max\_pool\_2d(convnet, 5)

    convnet = conv\_2d(convnet, 64, 3, activation='relu')

    convnet = max\_pool\_2d(convnet, 5)

    convnet = conv\_2d(convnet, 32, 5, activation='relu')

    convnet = max\_pool\_2d(convnet, 5)

    convnet = fully\_connected(convnet, 1024, activation='relu')

    convnet = dropout(convnet, 0.8)

    convnet = fully\_connected(convnet, num, activation='softmax')

    convnet = regression(convnet, optimizer='adam', learning\_rate = 0.001, loss='categorical\_crossentropy')

    model = tflearn.DNN(convnet, tensorboard\_verbose=1)

    return model

def load\_data(): #returns (X\_train,Y\_train,X\_test,Y\_test)

    dirs = getlabel()

    data = []

    label = []

    l = 0

    for i in dirs:

        filedirectory = os.path.join('Datadir',i)

        for k in range(1,41): #in range of the number of pics

            newdir = os.path.join(filedirectory,str(k)+'.jpg')

            image = cv2.imread(newdir, cv2.IMREAD\_GRAYSCALE)

            image = cv2.resize(image, (50,50), interpolation = INTER\_AREA)

            imagearray = numpy.array(image)

            imagearray.reshape(50,50,1)

            data.append(imagearray)

            label.append(l)

        l+=1

    return sklearn.model\_selection.train\_test\_split(data, label, test\_size=0.20, random\_state=42)

def createIDdir(ID):

    directory = 'Datadir'

    directory = os.path.join(directory,str)

    os.makdir(directory)

def getdir(ID):

    directory = 'Datadir'

    if (not os.path.isdir(directory)):

        os.mkdir(directory)

    directory = os.path.join(directory, str(ID))

    if (not os.path.isdir(directory)):

        os.mkdir(directory)

    return directory

#updatemodel()

Application:

from flask import Flask, render\_template, request, Response, redirect, url\_for, jsonify

from Camera import Camera

app = Flask(\_\_name\_\_)

app.config['SECRET\_KEY'] = 'secret!'

app.config['DEBUG'] = True

global feed\_status

global capture\_status

feed\_status = {}

capture\_status = 0

@app.route("/")

def index():

    return render\_template("login.html")

@app.route("/user")

def user():

    return render\_template("user.html")

@app.route("/sign\_in")

def signin():

    return render\_template("signin.html", name="notspecified")

@app.route("/sign\_up", methods=['POST'])

def signup():

    name = request.form.get("uname")

    print(name)

    return render\_template("signup.html", name=name)

def gen(camera, person):

    global feed\_status

    feed\_status[person] = 0

    print(feed\_status)

    while True:

        #print("going to get frame")

        status, frame = camera.getframe() #the difference between signup and signin is the obj provided

        #print(status)

        yield (b'--frame\r\n'

               b'Content-Type: image/png\r\n\r\n' + frame + b'\r\n\r\n')

        if status == 1:

            print(status)

            feed\_status[person] = 1

            print('setting up status to 1')

            break

@app.route('/sign\_up/api')

def image\_feed():

    name = request.args.get("name")

    cam = Camera(name)

    fobject = gen(cam, person=name)

    return Response(fobject,

                    mimetype='multipart/x-mixed-replace; boundary=frame')

@app.route('/sign\_up/status', methods=['POST'])

def image\_feed\_status():

    if request.method == "POST":

        json\_data = request.get\_json()

    name = json\_data[0]['msg']

    print("jason name:",name)

    print(feed\_status)

    if name in feed\_status:

        status = feed\_status[name]

    else:

        status = 0

    result = {'message' : status}

    return jsonify(result)

@app.route('/sign\_in/api')

def image\_capture():

    print("Inside here")

    cam = Camera("notspecified")

    fobject = gendetect(cam, person="notspecified")

    print("here")

    return Response(fobject,

                    mimetype='multipart/x-mixed-replace; boundary=frame')

def gendetect(camera, person):

    print("gendetect")

    global capture\_status

    while True:

        status, frame, ID, name = camera.detectface() #the difference between signup and signin is the obj provided

        print(status)

        yield (b'--frame\r\n'

               b'Content-Type: image/png\r\n\r\n' + frame + b'\r\n\r\n')

        if status == 1:

            capture\_status = ID

            print("found a person",name)

            break

@app.route('/sign\_in/status', methods=['POST'])

def img\_capture\_status():

    if request.method == "POST":

        json\_data = request.get\_json()

    name = json\_data[0]['msg']

    print("jason name:",name)

    print(capture\_status)

    # if name in feed\_status:

    #   status = feed\_status[name]

    # else:

    #   status = 0

    result = {'message' : capture\_status}

    return jsonify(result)

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(host='0.0.0.0', debug=True)