CONTROLLING A MOTOR WITH FACE RECOGNITION

ABSTRACT

Here we will be trying to operate a motor when a specific face is recognized, with the help of the Python Face Recognition library, OpenCV, and Arduino.

What is Face Recognition?

Face recognition is a way of identifying or confirming an individual’s identity using their face. Facial recognition systems can be used to identify people in photos, videos, or in real time.

Python world’s simplest face recognition library. Built using dlib’s state-of-the-art face recognition built with deep learning. The model has an accuracy of 99.38% on the Labelled Faces in the Wild benchmark.

What is OpenCV?

OpenCV is a Python open-source library, which is used for computer vision in Artificial intelligence, Machine Learning, face recognition, etc.

In OpenCV, the CV is an abbreviation form of computer vision, which is defined as a field of study that helps computers to understand the content of digital images such as photographs and videos.

What is Arduino?

Arduino is an open-source electronics platform based on easy-to-use hardware and software. Arduino boards are able to read inputs – a light on a sensor, a finger on a button, or a Twitter message - and turn them into an output activating a motor, turning on an LED, and publishing something online. You can tell your board what to do by sending a set of instructions to the microcontroller on the board. To do so you use the Arduino programming language (based on Wiring), and the Arduino Software (IDE), based on Processing.

How does it work?

1. Encode a picture using the HOG algorithm to create a simplified version of the image. Using this simplified image, find the part of the image that almost looks like a generic HOG encoding of a face.
2. Figure out the pose of the face by finding the main landmarks in the face. Once we find those landmarks, use them to warp the image so that the eyes and mouth are centered.
3. Pass the centered face image through a neural network that knows how to measure the features of the face. Save those 128 measurements.
4. Looking at all the faces we've measured in the past, see which person has the closest measurements to our face's measurements. That's our match!

Face Recognition – Step by Step

Step 1: Finding all the Faces

The first step in our pipeline is face detection. Obviously, we need to locate the faces in a photograph before we can try to tell them apart!

We're going to use a method invented in 2005 called Histogram of Oriented Gradients — or just HOG for short.

To find faces in an image, we'll start by making our image black and white because we don't need colour data to find faces.

Then we'll look at every single pixel in our image one at a time. For every single pixel, we want to look at the pixels that directly surround it:

Our goal is to figure out how dark the current pixel is compared to the pixels directly surrounding it. Then we want to draw an arrow showing in which direction the image is getting darker.

If you repeat that process for every single pixel in the image, you end up with every pixel being replaced by an arrow. These arrows are called gradients and they show the flow from light to dark across the entire image.

If we analyze pixels directly, really dark images and really light images of the same person will have totally different pixel values. But by only considering the direction that brightness changes, both really dark images, and really bright images will end up with the same exact representation. That makes the problem a lot easier to solve!

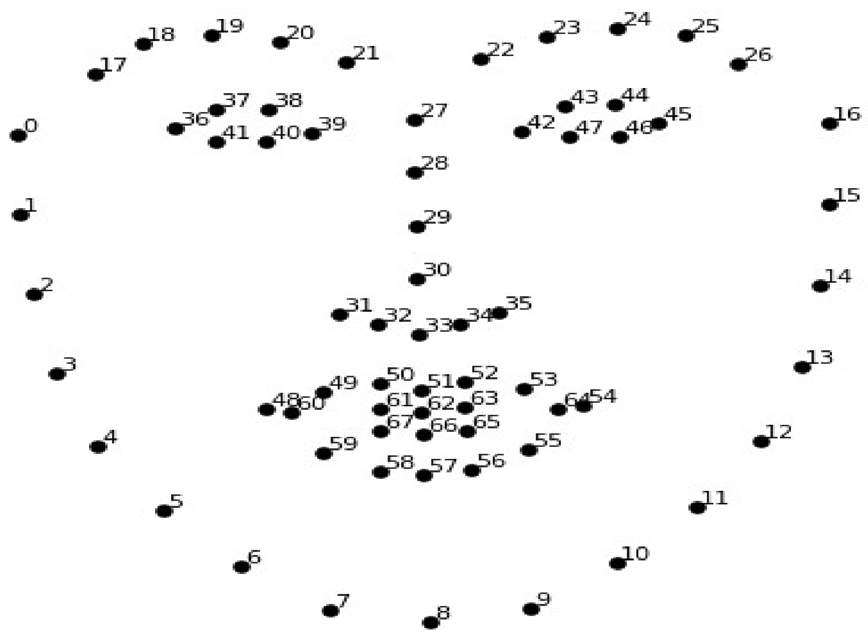
But saving the gradient for every single pixel gives us way too much detail. We end up missing the forest for the trees. It would be better if we could just see the basic flow of lightness/darkness at a higher level so we could see the basic pattern of the image.

To do this, we'll break up the image into small squares of 16x16 pixels each. In each square, we'll count up how many gradients point in each major direction (how many point up, point up-right, point right, etc…). Then we’ll replace that square in the image with the arrow directions that were the strongest.

The end result is we turn the original image into a very simple representation that captures the basic structure of a face in a simple way.

Step 2: Posing and Projecting Faces

The basic idea is we will come up with 68 specific points (called landmarks) that exist on every face — the top of the chin, the outside edge of each eye, the inner edge of each eyebrow, etc. Then we will train a machine learning algorithm to be able to find these 68 specific points on any face.



Step 3: Encoding Faces

The face_recognition API generates face encodings for the face found in the images. A face encoding is basically a way to represent the face using a set of 128 computer-generated measurements. Two different pictures of the same person would have similar encoding and two different people would have totally different encoding.

After all the face encodings are generated, Support Vector Classifier (SVC) with scikit-learn is trained on the face encodings along with their labels from all the known faces in the training directory. Finally, the API detects all the faces in the test image you provide and the trained SVC predicts all the known faces in the test image.

Step 4: Finding the person’s name from the encoding

This step is actually the easiest step in the whole process. All we have to do is find the person in our database of known people who has the closest measurements to our test image.

After finding the closest image get the name of that image using indexing.

Step 5: Pushing signal to Arduino to run motor

Here we will be using the Pyserial module of python to push the data from our python code to Arduino.

The minimum distance required to found a match from encoded image is 0.60, to make the recognition process more strict we will fix it to 50 (The motor will work only if the face distance is less than .50, it will give us more security)

THE CODE:

1. Python code

   - Import Necessary Libraries in Python

```
import cv2
import numpy as np
import face_recognition
import os
from datetime import datetime
import serial
```

   - Setting up Images path For loading all images

```
path = 'ImageBasics'
images = []
classNames = []
nameList = []
myList = os.listdir(path)

# find the name of the person from image name and add images to a list.

for cl in myList:
    curImg = cv2.imread(f'{path}/{cl}')
    images.append(curImg)
    classNames.append(os.path.splitext(cl)[0])
```

- Finding the Face Encodings

```
def findEncodings(images):
    encodeList = []
    for img in images:
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        encode = face_recognition.face_encodings(img)[0]
        encodeList.append(encode)
    return encodeList

def captureScreen(bbox=(300,300,690+300,530+300)):
    capScr = np.array(ImageGrab.grab(bbox))
    capScr = cv2.cvtColor(capScr, cv2.COLOR_RGB2BGR)
    return capScr
encodeListKnown = findEncodings(images)
print('Encoding Complete')
```

- Use Video Camera to capture images/faces

```
cap = cv2.VideoCapture(0)

while True:
    success, img = cap.read()
    #img = captureScreen()
    imgS = cv2.resize(img,(0,0),None,0.25,0.25)
    imgS = cv2.cvtColor(imgS, cv2.COLOR_BGR2RGB)

# open video capture and detect a face, find and compare its encodings
by the distance, and if the distance is within the min range, show
recognition
```

- Face Locations and Face Distances after comparison

```
facesCurFrame = face_recognition.face_locations(imgS)
encodesCurFrame =
face_recognition.face_encodings(imgS,facesCurFrame)

for encodeFace,faceLoc in zip(encodesCurFrame,facesCurFrame):
matches =
face_recognition.compare_faces(encodeListKnown,encodeFace)
faceDis =
face_recognition.face_distance(encodeListKnown,encodeFace)
#print(faceDis)
matchIndex = np.argmin(faceDis)
```

- Compare Face Encoding Differences for Threshold and finding matches

```
if matches[matchIndex]:
            name = classNames[matchIndex]
            print(name)

            y1,x2,y2,x1 = faceLoc
            y1, x2, y2, x1 = y1*4,x2*4,y2*4,x1*4
            cv2.rectangle(img,(x1,y1),(x2,y2),(0,255,0),2)
            cv2.rectangle(img,(x1,y2-
35),(x2,y2),(0,255,0),cv2.FILLED)
cv2.imshow('Webcam',img)
```

- Pushing data to Arduino

```
    distance=round(faceDis[0],2)
    print(distance)
    if (distance < 0.50):  # if distance less than 0.50
            Arduino_Serial.write(str.encode('1'))  # send 1 to
arduino
            print("MOTOR IS RUNNING!!!")

    if (distance > 0.50):  # if distance is greater than 0.50
            Arduino_Serial.write(str.encode('2'))  # send 0 to
arduino
            print("MOTOR TURNED OFF!!!")
```

- Close allwindows

```
    cv2.imshow('Webcam',img)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
cap.release()
```

2. Arduino code

```
const int led=13; # initialize the pin
int value=0;
int count=0; to stop rotation face a specific time interval
void setup()
   {
      Serial.begin(9600);
      pinMode(led, OUTPUT);
      digitalWrite (led, LOW);
      Serial.println("Connection established...");
   }

void loop()
   {
     while (Serial.available())
```

```
            {
               value = Serial.read();
            }
         count=0;
         if (value == '1' && count<=10)
         {
            digitalWrite (led, HIGH);
            count++;
         }

         else if (value == '2')
            digitalWrite (led, LOW);
       }
```