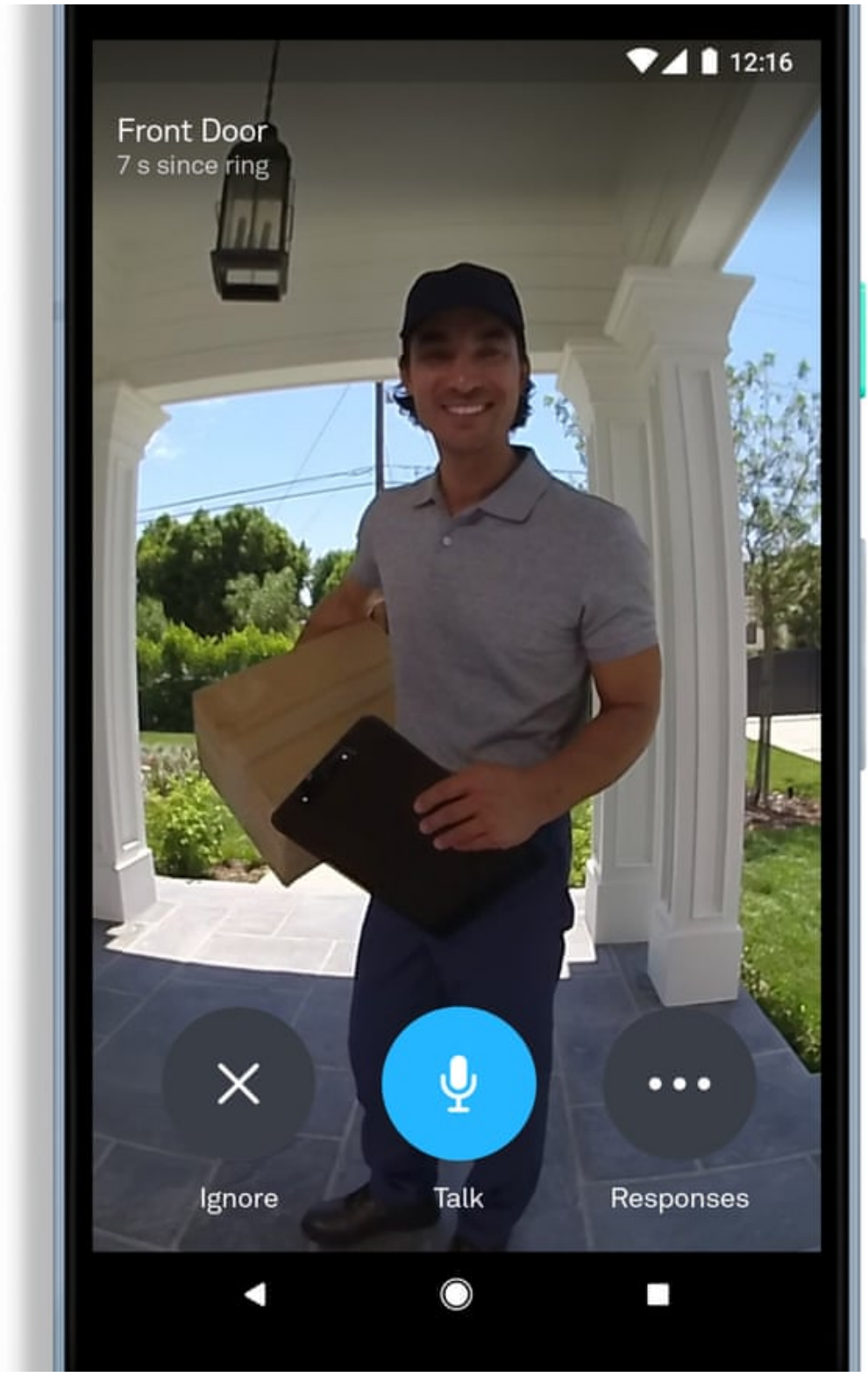

12:16
Front Door
7 s since ring
Ignore
Talk
Responses

Doorbell camera Face Detection

Aakash Chaudhary

CSC 482

Table of Contents

Abstract

Face detection by Computer frameworks has become a significant field of intrigue. Face identification calculations are utilized in a wide scope of uses, for example, security control, video recovering, biometric signal handling, human Computer interface, face detection and picture database the executives. In any case, it is difficult to build up a total strong face identifier because of different light conditions, face sizes, face directions, background and skin hues. In this, I propose a continuous face detection method. We are utilizing the advances accessible in the Open-Computer Vision (OpenCV) library and technique to actualize them utilizing Python. The approach is depicted including pictures of our yield.

Problem statement

I need to recognize individuals I've never seen before, so I won't start with a list of known appearances to check against. Yet, regardless of whether I don't have the foggiest idea what someone's identity is, I can in any case use face detection to recognize them as a similar individual each time they return

Objective

The goal of this paper, which I believe I have come to, was to build up an Algorithm for face detection that is quick, vigorous, sensibly simple and efficient with a moderately straightforward and straightforward algorithms and methods. The instance provided in this paper are real time and taken from our surrounding.

History of Face-Detection

The concept of Face-Detection is as old as computer vision. face detection has consistently stayed a significant focal point of research due to its non-intrusive nature and since it is individuals' essential strategy for individual identification. For all I know, the most well-known early case of a face detection framework is due to kohonen, who showed that a simple neural net could perform face detection for adjusted and standardized face pictures. The sort of network he utilized processed a face description by approximating the eigenvectors of the face picture's autocorrelation network; these eigenvectors are currently known as 'eigenfaces.' Kohonen's framework was not practical achievement, be that as it may, in light of the requirement for exact arrangement and standardization. In following years numerous researchers attempted face detections plans dependent on edges, feature distances, and other neural net methodologies. While a few were fruitful on little databases of adjusted pictures, none effectively tended to the more realistic issue of huge databases where the area and size of the face is obscure. Kirby and Sirovich (1989) later presented a mathematical manipulation which made it simple to legitimately compute the eigenfaces and appeared that less than 100 were required to precisely code carefully aligned and standardized face pictures. Turk and Pentland (1991) at that point exhibited that the residual error when coding utilizing the eigenfaces could be utilized both to detect faces in jumbled regular symbolism, and to decide the exact location and size of appearances in a picture. They at that point exhibited that by coupling this strategy for identifying and localizing faces with the eigenface recognition technique, one could accomplish solid, real time detection of faces in a minimally obliged environment. This exhibition that simple, real time pattern detection procedures could be consolidated to make a helpful framework started a blast of enthusiasm for the subject of face detection.

Methodology

Requirement

knowledge on NumPy and Matplotlib is fundamental before dealing with the ideas of OpenCV. Ensure that you have the accompanying Library installed and running before introducing OpenCV.

1. Python
2. NumPy
3. Matplotlib

Method

Here's the means by which we can follow individuals I've never seen utilizing face encodings:

1. Each time I see a face, calculate a face encoding for it utilizing our face detection model.

2. Check all the faces I've seen beforehand to check whether any of their face
encodings appear to be a close to coordinate for this new face.

3. On the off chance that the new face coordinates a face encoding I have seen previously, I
realize I am taking a gander at a similar individual once more! So, I can refresh the count that anonymous individual has visited.

4. In the event that the new face encoding doesn't match any recently observed appearances, make another record for it in our list of recently observed faces with a visit tally of 1.

As I see an ever-increasing number of countenances, the list of Seen faces will develop. In the end I'll have a database of everybody who has ever come to our doorway even despite the fact that I won't know their names.

Face Detection using OpenCV

I have used Jupyter notebook for this project. For using OpenCV first need to import the OpenCV library, NumPy library, matplotlib library

```
import face_recognition
import cv2
import matplotlib.pyplot as plt
import numpy as np
```

Then, I created some variable to store the data of the people who came in front to my door camera. All of these variables will be my database of known visitor who have come by inform of my door previously.

```
known_face_encodings = []
known_face_metadata = []
```

After, I create a function to save known face data.

```
def save_known_faces():
    with open("known_faces.dat", "wb") as face_data_file:
        face_data = [known_face_encodings,known_face_metadata]
        pickle.dump(face_data, face_data_file)
        print("Known faces backed up to disk.")
```

And the load function works the same in the reverse. Every time my program recognizes new appearances, I will call a function to add into my known database.

```
def load_known_faces():
    global known_face_encodings, known_face_metadata
    try:
        with open("known_faces.dat", "rb") as
        face_data_file:
            known_face_encodings, known_face_metadata = pickle.load(face_data_file)
            print("Known faces loaded from disk.")
    except FileNotFoundError as e:
        print("No previous face data found.")
    pass
def register_new_face(face_encoding, face_image):
    known_face_encodings.append(face_encoding)
    known_face_metadata.append({"first_seen": datetime.now(),
                                "first_seen_this_interaction": datetime.now(),
                                "last_seen": datetime.now(),"seen_count": 1,
                                "seen_frames": 1,"face_image": face_image,11 })
```

To begin with, I am storing the face encoding that represent to the face in a list. At that point, I am storing a coordinating dictionary of information about the face in a second list. I'll utilize this to follow the time I previously observed the individual, to what extent they've been sticking around the camera as of late, how frequently they have visited home, and a little picture of their face. I additionally need a helper function to check if an unknown face is as of now in our face database or not.

Next, I am doing a couple of significant things here:
1. Utilizing the face_recogntion library, I check how similar the unknown face is to every single past guest. The face_distance() work gives us a numerical estimation of comparability between the unknown face and every known face—the littler the number, the more similar the faces.

2. On the off chance that the face is fundamentally the same as one of our known guests, we expect they are a recurrent guest. All things considered, I update their "last observed" time and addition the occasions we have seen them in a frame of video.

3. At long last, if this individual has been found before the camera in the last five minutes, I expect they are still here as a feature of a similar

visit. Else, I accept this is another visit to home, so I'll reset the time stamp following their latest visit.

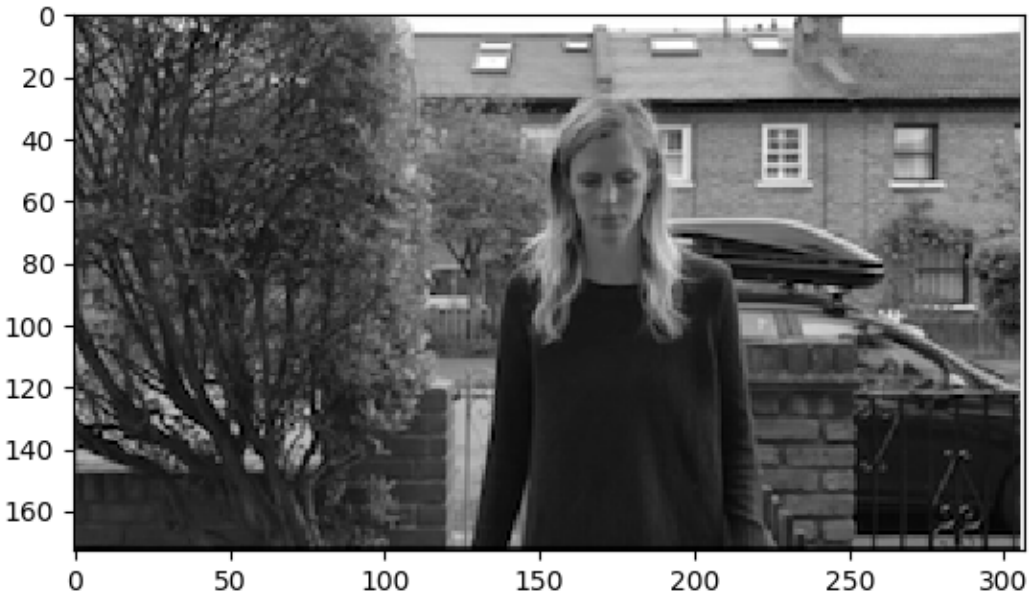
The remainder of the program is the main loop—a perpetual circle wherein I get a frame of video, search for faces in the picture, and procedure each face we see. It is the fundamental heart of the program. We should look at it.

```
def main_loop():
    video_capture = cv2.VideoCapture("door.mp4")
```

It will take frame from the store door camera
Every time we get a frame of camera video, we'll additionally contract it to 1/4 size. This will make the face identification process run quicker to the detriment of as it were identifying bigger faces in the picture. Be that as it may, Because, I am building a Front door camera that just perceives individuals close to the FrontDoor, that shouldn't be a issue. We likewise need to manage the way that OpenCV pulls pictures from the camera with every pixel put away as a Blue-Green-Red incentive rather than the standard request of Red-Green-Blue. Before we can run face acknowledgment on the picture, we have to reorder the shading channels. NumPy Achieve this. it stats that small_frame[:, :, ::- 1] tells NumPy to peruse the color channels in reverse order, which helpfully swaps them from Blue-Green-Red to Red-Green-Blue.

Presently, we can detect faces in the picture and convert each face into a face encoding. That lone takes two lines of code.

I added a list of previous guest at the right corner video with the count our camera has detected their faces.

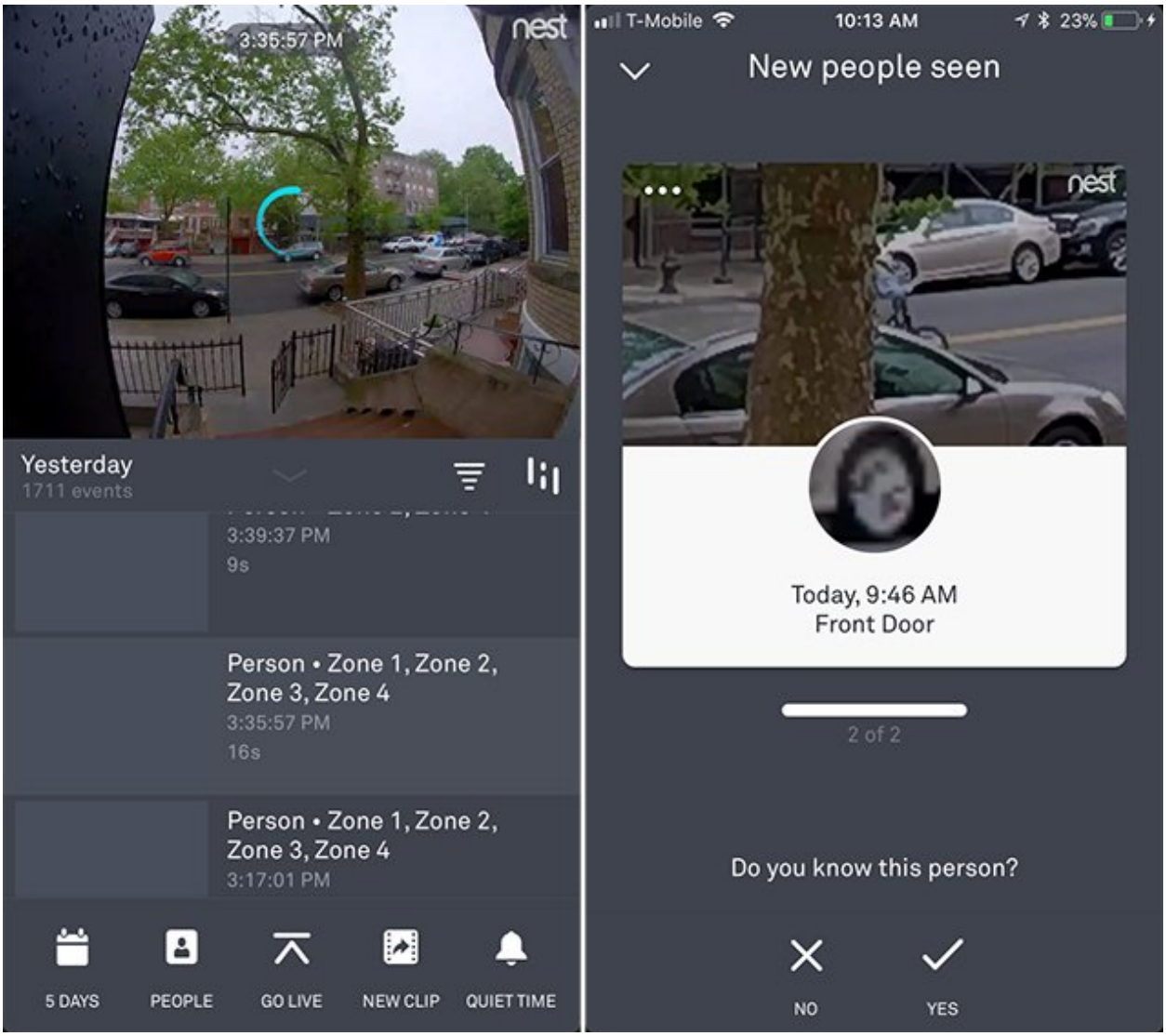


Conclusion

At the point I run the code, I have seen a video window spring up on work area. At whatever point another individual stride before the camera, it is enrolling their face and begin following to what extent they have been close to the doorway. In this event that a similar individual leave and returns over five minutes after the fact. for this work I used an example video from YouTube. The application will consequently save data about everybody it sees to a file called known_faces.dat. At the point when I run the program once more, it is utilizing that information to recollect past guests. if I want to clear this list of known appearances. I just quit the program and deleted that file

Future work

On this project we create an application than we can send a text notification whenever a new person will be detected in our doorbell camera



Source ~~ Google

References

1. Hsu, R. L., Abdel-Mottaleb, M., Jain, A. K. (2002). Face detection in color images. IEEE transactions on pattern analysis and machine intelligence, 24(5), 696-706.

2. Hjelm_as, E., Low, B. K. (2001). Face detection: A survey. Computer vision and image understanding, 83(3), 236-274.

3. Viola, P., Jones, M. (2001, July). Robust real-time face detection. In null (p. 747). IEEE.

4. Bartlett, M. S., Littlewort, G., Fasel, I., Movellan, J. R. (2003, June). Real Time Face Detection and Facial Expression Recognition: Development and Applications to Human Computer Interaction. In 2003 Conference on computer vision and pattern recognition workshop (Vol.5, pp. 53-53). IEEE.