
# FACE DETECTION AND RECOGNITION REPORT

Instructor: Dr.Cao Tien Dung

Course: Introduction to Computer Science

Tan Tao University

Group Members:

1. Nguyễn Đức Nam
2. Trần Triệu Tuân
3. Huỳnh Thị Mỹ Hạnh

# Table of Contents:

# Abstract

Face detection has attracted great attention because it has many applications in computer vision communication and automatic control systems. Face detection is a method to detect a face from an image that has some attributes in that image. It is necessary to study face detection, expressive recognition, face tracking, posture estimation. By giving a single image, the challenge is to detect the face from that image. Face recognition is a challenging task as the face is not stiff and it changes in size, shape, color, etc. Face detection becomes a more difficult task when the provided image is unclear and obscured by anything else and there is no proper lightning bolt, not facing the camera, etc. However, it is less robust for a fingerprint or retinal scan. This report describes a small facial recognition and detection project implemented for the visual perception and self-control module through the CS111 course. In this paper, the viola Jones algorithm is used for face detection and key component analysis for facial recognition.

## Keywords:

DCT, Viola jones algorithm, Face Recognition, Face Detection


# I. Introduction

Image processing is an image processing technique or algorithm for compressing images, enhancing images, or extracting any useful information from an image. For image processing, there are two methods, digital and analog. Digital image processing is processed by a digital computer using a mathematical model and computer algorithms to process a digital image and a digital image. The same image processing is used for hard copies such as photos and prints. Face recognition is the realm of image processing.

In recent years, facial recognition has been studied due to the widespread use of human-computer interactions.f. Video surveillance has several phases:

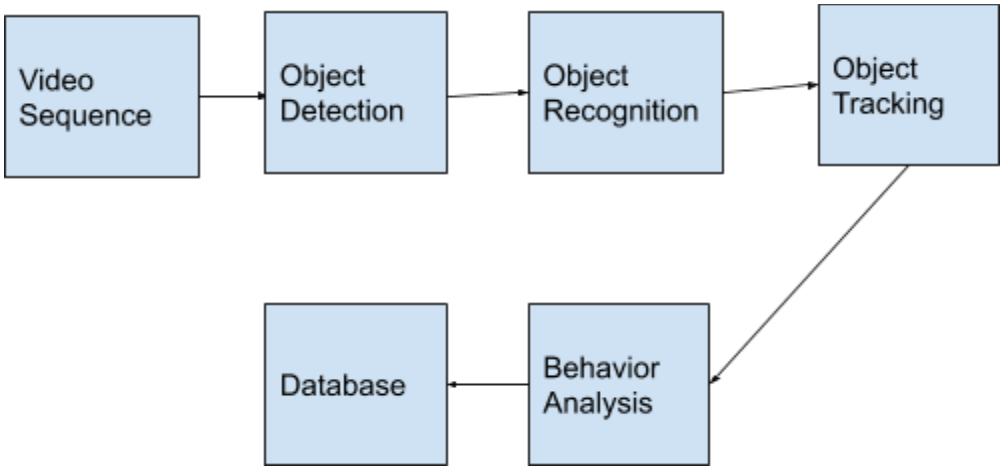


*Fig. 1. Stage in Video Management*

In this research field some terms regarding face is “facial feature detection” means to locate the human face features such as eye, nose, lips, chin, eyebrows, mouth, ears etc.“Face authentication” is to verify the identity of a person. “Face tracking” is to track a face location in real time. “Expression recognition” is to recognize the facial expression of a face, happy, sad, surprising etc. “face localization” is to identify the location of a face.

# II. Design

This is a Video Capture application that helps users record videos, take snapshots and more specifically detect a user's face as well as recognize who they are if that person's face is stored in the database. Hence it is very useful in businesses to check for employee attendance as well as in security. The App is shown in the below and explains each part that our team has completely done.

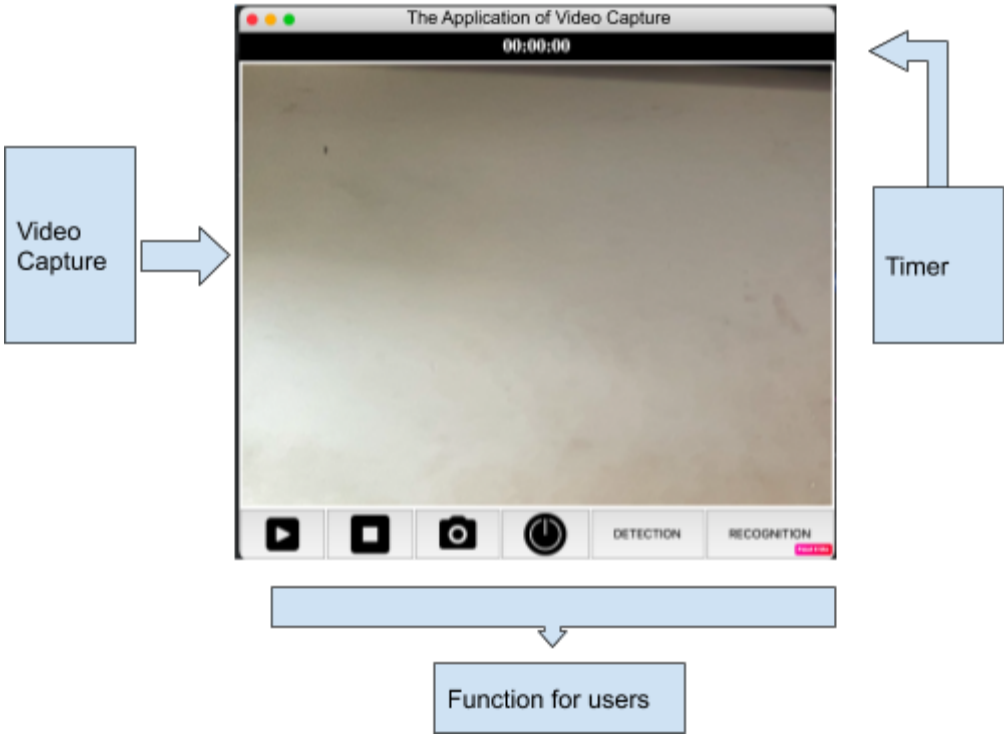



*Fig. 2. The Application of Video Capture*

For the convenience of explaining the functions of the application and also demonstrating the association between them, we created a flowchart between them so that the readers could easily see their correlations.

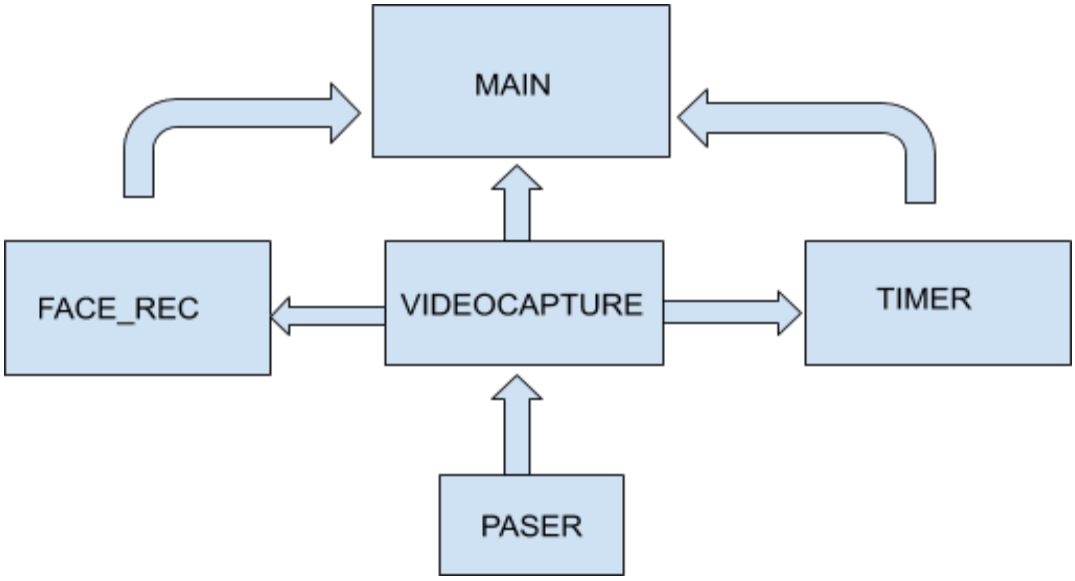



*Fig. 3. The flowchart between modules*

1. Module **VIDEOCAPTURE**: to open a webcam and display video in real time on GUI.

   **Solution**: we use the Open-CV library because Open-CV is a stronger library, it has a big community to exchange and learn with others in the community. Open-CV uses low RAM(approx 60-70mb) and It is portable as OpenCV can run on any device that can run C. By using it, one can process images and videos to identify objects, faces, or even handwriting of a human. When it is integrated with various libraries, such as Numpuy, Python is capable of processing the OpenCV array structure for analysis. To Identify image pattern and its various features we use vector space and perform mathematical operations on these features. So there are reasons why we chose Open-CV for my project.

★ Creating the **class Videocapture** in this module

- Using **cv2.Videocapture()** to open a webcam. To capture a video, you need to create a VideoCapture object. Its argument can be either the device index or the name of a video file. Device index is just the number to specify which camera. Normally one camera will be connected (as in our case). So we simply pass 0 (or -1). You can select the second camera by passing 1 and so on.

- Using **cv2.VideoWriter()** to record video. We should specify the output file name (for example, video.avi). Then we should specify the FourCC code (details in the next paragraph). Then, the number of frames per second (fps) and the frame size are transferred. And finally the isColor flag. If it is True, the encoder expects color frames, otherwise it works with grayscale frames.

- Creating a function to get frames, get output video and release the video source by **cv2.destroyAllWindows().** For the function **cv2.destroyAllWindows()** that is written in the **"OpenCV"** library, It simply destroys all the windows we created. If you want to destroy any specific window, use the function cv2.destroyWindow() where you pass the exact window name as the argument.

2. Module **FACE_REC**: To get encode of faces and compare encode of face between face in database and face in real time.

**Solution:** we use face_recognition library to do that because Recognize and manipulate faces from Python or from the command line with the world's simplest face recognition library.

- Built with dlib's modern face recognition built with deep learning methods by C++ (so it runs very fast to compile).
- The model has an accuracy of 99.38% above
- Faces are labeled in the Wild standard.
- This also provides a simple **face_recognition** command line tool that allows you to do facial recognition on an image directory from the command line!

★ Creating the **functions** to implement in this module:

- Using **face_recognition.face_encodings(face)[0]** to get encode faces. For example, given an image, return the 128-dimension face encoding for each face in the image.

- Using **face_recognition.face_locations(img)** to return an array of bounding boxes of human faces in an image. This is a list of tuples of found face locations in css (top, right, bottom, left) order.

- Using **face_recognition.face_distances(face_encodings, face_to_compare)** to return a numpy ndarray with the distance for each face in the same order as the 'faces' array. For instance,

given a list of face encodings, compare them to a known face encoding and get a euclidean distance for each comparison face. The distance tells you how similar the faces are.

- Using **face_recognition.compare_faces()** to compare a list of face encodings against a candidate encoding to see if they match. It returns a list of **True/False** values indicating which known face_encodings match the face encoding to check.

3. Module **TIMER:** to display a clock count time when users press the start button and stop when users press the stop button.

   **Solution:** we use datetime library and tkinter and the reason why we use datetime, tkinter library.

   - Datetime is a good library support for developers
   - it's very simple to learn and to use.
   - The second reason, tkinter have many advantages for developers such as easy to accessibility,
   - The layered approach used in designing Tkinter gives Tkinter all of the advantages of the TK library
   - portability : Python scripts that use Tkinter do not require modifications to be ported from one platform to the other. This gives it a great advantage over most competing libraries, which are often

restricted to one or two platforms. Moreover, Tkinter will provide the native look-and-feel of the specific platform it runs on.

- Availability: Tkinter is now included in any Python distribution. Therefore, no supplementary modules are required in order to run scripts using Tkinter

4. Module **MAIN:** Import **class Videocapture** from module **VIDEOCAPTURE**, import class ElapsedTimeClock from module **TIMER**, import **functions** in module **FACE_REC,** and Creating buttons are **start, stop, snapshot, exit, detection, recognition.**

   **Solution:** we use tkinter, pygame to do that.

   - Using **tk.Canvas** to create the main frame.
   - Using **tk.PhotoImage** to put an image in the button.
   - Using **tk.Button** to create buttons **start, stop, snapshot, exit, detection, recognition.**

5. Module **PASER:**

   The **argparse** module makes it easy to write user-friendly command line interfaces. The program determines the arguments it requires, and **argparse** finds a way to parse those arguments out of **sys.argv**. The **argparse** module also automatically generates help and usage messages and throws errors when the user gives the program invalid arguments.

- Using **argparse.ArgumentParser(description="...")** that will hold all the information necessary to parse the command line into Python data types.

- Filling an **ArgumentParser** with information about program arguments is done by making calls to the **add_argument()** method. Generally, these calls tell the **ArgumentParser** how to take the strings on the command line and turn them into objects. This information is stored and used when **parse_args()** is called.

- **ArgumentParser** parses arguments through the **parse_args()** method. This will inspect the command line, convert each argument to the appropriate type and then invoke the appropriate action. In most cases, this means a simple **Namespace** object will be built up from attributes parsed out of the command line.

# III. Implementation

## A. Face Detection

In this paper viola jones algorithm is used for face detection and the process of face detection for a single image different steps are used, these are as follows:

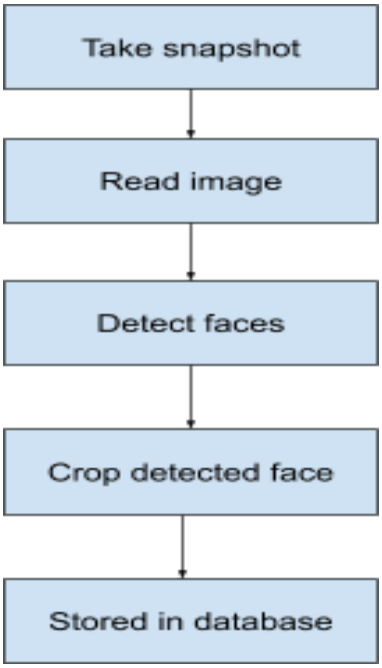



*Fig. 4. Process of Face Detection*

Following the flow chart, we have to do:

1) Now we create the cascade and initialize it with our face cascade. This loads the face cascade into memory so it’s ready for use. Remember, the cascade is just an XML file that contains the data to detect faces.

```
self.face_cascade=cv2.CascadeClassifier('haarcasde/haarcascade_frontalface_default.xml')
```

★ Let's take a closer look at what Haar Cascade is:

Haar-like method, which enables the detector to be fast by quickly rejecting windows that are clearly not faces, is known as the method using features or specific structure of an object to detect it from an image. A Haar-like feature considers neighboring rectangular regions at the specific location in a detection window, sums up the pixel intensities in each region and calculates the difference between these sums.

This difference is then used to categorize subsections of an image. When the system detects an object, a window is moved over the image. At each time the window is moved, a current location is given back with positive or negative results. If the result is negative, the window will be moved to other locations in the image. However, if the result is positive, this means that the location is the object that needs to be detected, the system will automatically move to the next step of processing.

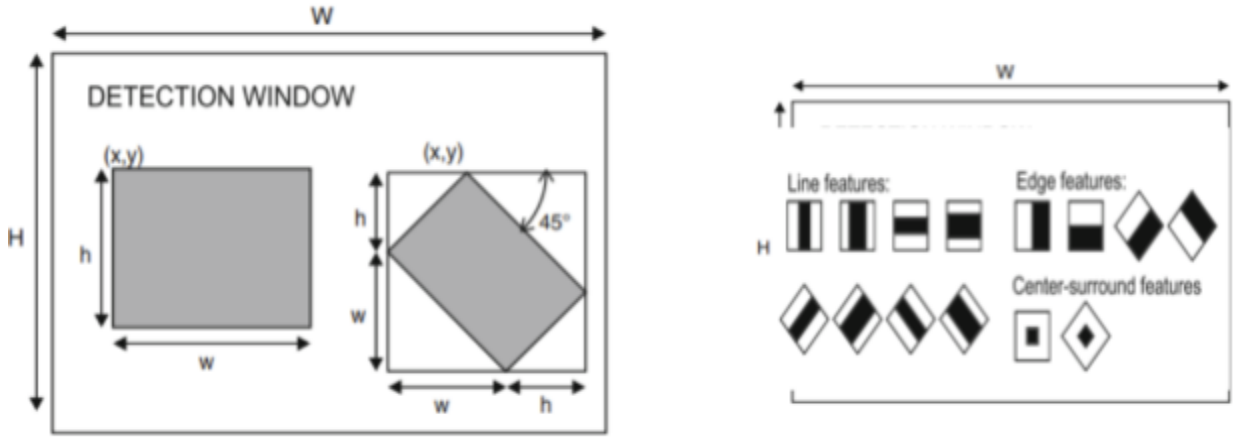



*Fig. 5. The explanation of Haar-cascade file*

2) Capture image: To give input image to be read first Image is captured and stored into a database. Other images can be used which are stored already into the dataset.

```
class VideoCapture:
    def __init__(self, video_source=0):

        # Open the video source
        self.vid = cv2.VideoCapture(video_source)
        if not self.vid.isOpened():
            raise ValueError("Unable to open video source", video_source)
```

```
# To get frames
    def get_frame(self):
        if self.vid.isOpened():
            ret, img = self.vid.read()
            if ret:
                # Return a boolean success flag and the current frame converted to
cv2.COLOR_RGB2BGR
                return (ret, cv2.cvtColor(img, cv2.COLOR_RGB2BGR))

            else:
                return (ret, img)
        else:
            return (ret, None)
```

3) Reading pictures: real time images after being captured will be used with imwrite function and stored in the database.

```
def snapshot(self):
        # Get a frame from the video source

        ret, frame=self.vid.get_frame()

        # print(classify_face(frame))

        if ret:

cv2.imwrite("snapshot/IMG-"+time.strftime("%d-%m-%Y-%H-%M-%S")+".jpg",cv2.cvtColor(frame,cv2.
```

```
COLOR_RGB2BGR))
```

4) Face detection: The main goal of this step is to locate the face in its place in the image and detect it. Also create a border around the face by defining the background image. It works by detecting feature points and edges using the following equations.

```
if ret:

        # get a rectangle around face when click Detection
        if self.detect:
            gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
            faces = self.face_cascade.detectMultiScale(gray, 1.1, 4)

            for (x, y, w, h) in faces:

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

        self.photo = PIL.ImageTk.PhotoImage(image=PIL.Image.fromarray(frame))
        self.canvas.create_image(0,0, image=self.photo, anchor=tk.NW)
```

## B. Face Recognition

Face recognition is one of the most successful image analysis and understanding applications and has received a lot of attention in recent years. Face recognition is a computer-based application that detects the faces of different people for authentication, and more. It has separate apps with fingerprint and iris recognition. There are various successful techniques

proposed to date such as Comprehensive and Discrete Cosine Transformations (DCT). In this paper, we aim for a simple and fast facial recognition system based on feature extraction using Key Ingredient Analysis (PCA) which is a classic and successful method for Size reduction and the discrete cosine transform (DCT) paper is a well-known compression technique. The description of the face recognition process is here, and the face recognition process comprises the following steps:

### Step 1: Finding all image

The first step in our process is face recognition. Obviously we need to determine the position of the faces in an image before we can distinguish them! Face detection is a great feature of the camera. When the camera can select faces automatically, it can ensure that all faces are in focus before taking a photo. But we'll use it for a different purpose - find areas of the image that we want to move on to the next step in our process. It is important that the photos are in the same lighting conditions and the right face is in each photo. Also, the image used in this method must contain the same number of pixels and be grayscale.

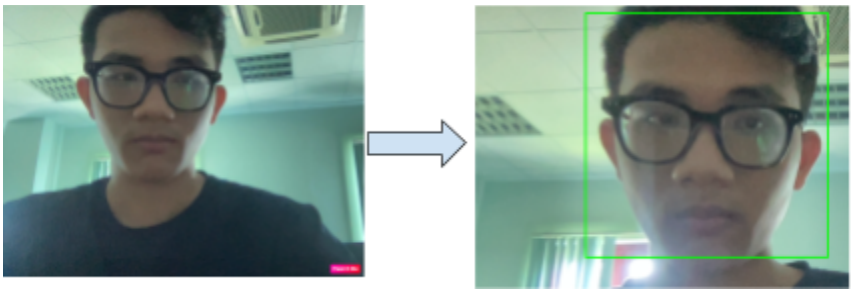


*Fig. 6. Determine the position of the faces in image*

## Step 2: Encoding Face

There are many ways to encode a face as used in the Discrete Cosine Transform (DCT) algorithm.

The algorithm use equation: $$v(k) = \alpha(k) \sum_{n=0}^{N-1} u(n) \cos\left(\frac{(2n+1)\pi k}{2N}\right)$$ $$0 \le k \le N-1$$ to calculate the distance between eyes, mouth, nose, height of face, width of face and average color of face by vectors feature. From the vectors feature into matrix vectors distance so the matrix shows complete features of the face . Different faces have different matrices so based on that to face recognition.

$$R = \begin{bmatrix} 1 & \rho & \rho^2 & .. & \rho^{N-1} \\ \rho & 1 & \rho & .. & \rho^{N-2} \\ . & . & . & .. & . \\ . & . & . & .. & . \\ \rho^{N-1} & \rho^{N-2} & . & .. & 1 \end{bmatrix}$$

$\rho \equiv$ correlation coeff.

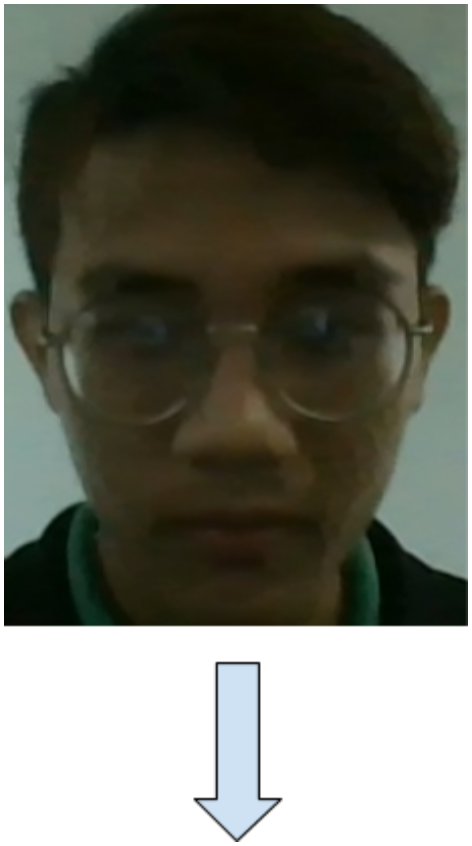
$|\rho| < 1$

(*this is the matrix vector distance of the face)*

Example:

| **Height of face(cm)** | **Width of face(cm)** | **Average color of face(R, G, B)** | **Width of lips(cm)** | **Height of nose(cm)** |
|---|---|---|---|---|
| 23.1 | 15.8 | (255,224,189) | 5.2 | 4.4 |

Our image now becomes a vector that could be represented as **[23.1, 15.8, 255, 224, 189, 5.2, 4.4]**



**[-0.23,-0.56,-0.76,....,-0.27]**

*Fig. 7. The vector features*

Each image is joined to create a vector, creating a 1 × n2 matrix. All images in the data set are stored in a single matrix that creates a matrix with columns corresponding to the number of images. The matrix is averaged (normalized) to get an average human face. By subtracting the average faces for each image vector, the distinct characteristics of each

face are calculated. In the resulting matrix, each column represents the difference of each face compared to the average human face.

In our project, we used face_recognition library to get encode of the face:

The first step: we go to the library which contains the database and we use loops to read all images in the database.

```
for dirpath, dnames, fnames in os.walk("./faces"):
        for f in fnames:
            if f.endswith(".jpg") or f.endswith(".png"):
```

Next step: we get encoding by face_recognition library:

```
face = fr.load_image_file("image/" + f)

        encoding = fr.face_encodings(face)[0]
        encoded[f.split(".")[0]] = encoding
```

The result:

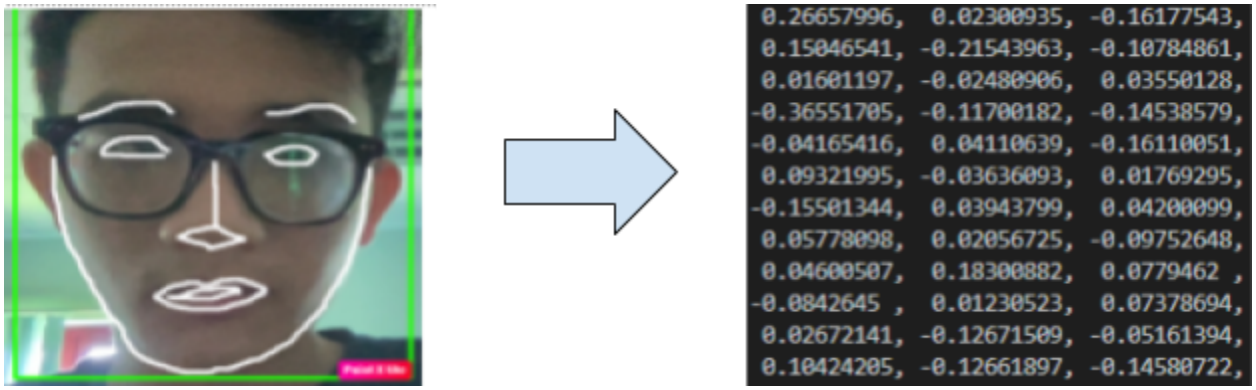



*Fig.8. this is the matrix vectors distance*

## Step 3: Finding the person's name from the encoding

This last step is actually the easiest step in the whole process. All we have to do is find people in our database of known people whose measurements are closest to our test image. You can do that using any of the basic machine learning taxonomy algorithms. No fancy deep learning tricks required. We will be using the Discrete Cosine Transform (DCT) algorithm, but many of the sorting algorithms might work. Following these code to see what they work:

```
def classify_face(img):
    # get encoding and contain in list
    faces = get_encoded_faces()
    faces_encoded = list(faces.values())
    known_face_names = list(faces.keys())

    # Returns an array of bounding boxes of human faces in a image
    face_locations = face_recognition.face_locations(img)
    unknown_face_encodings = face_recognition.face_encodings(img, face_locations)

    face_names = []
    for face_encoding in unknown_face_encodings:

        # See if the face is a match for the known face(s)
        matches = face_recognition.compare_faces(faces_encoded, face_encoding)

        name = "Unknown"

        # use the known face with the smallest distance to the new face
        face_distances = face_recognition.face_distance(faces_encoded, face_encoding)
        best_match_index = np.argmin(face_distances)
```

```
if matches[best_match_index]:
    for i in face_distances:
        if i <= 0.44:
            name = known_face_names[best_match_index]

face_names.append(name)
return face_names
```

As you can see in the code above, we use function **argmin** to measure the nearest between 2 images ( one in the **database** and other in the **frame**).

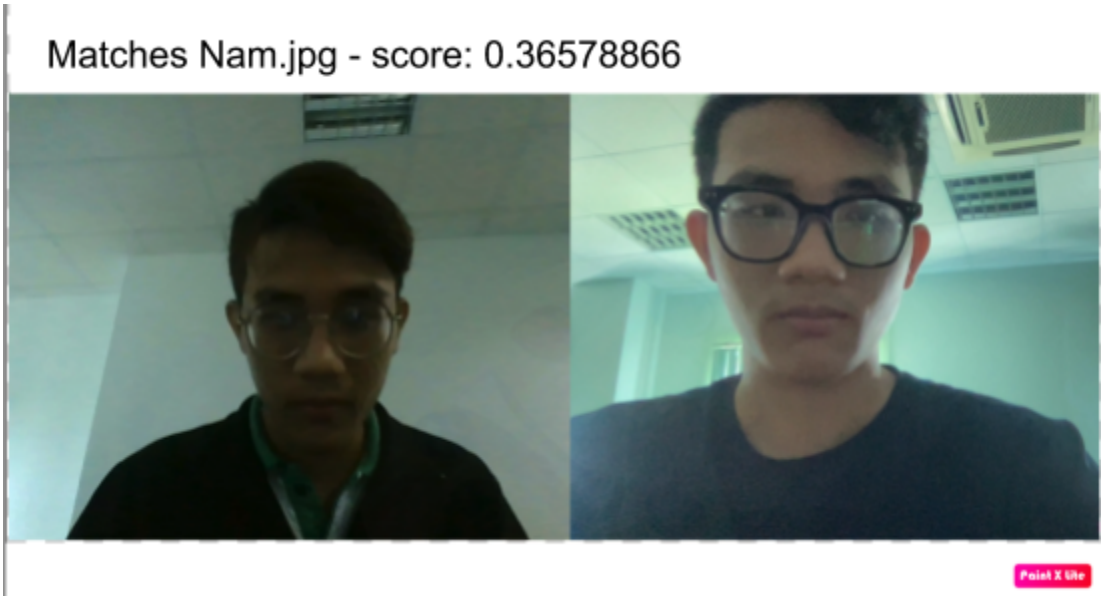



*Fig. 9. Compare real-time face with database*

All we need to do is train a classifier that can take measurements from the new test image and indicate who is the best match and resubmit that person's name on the Terminal. Running this classifier takes milliseconds. The result of

the classifier is the person's name! To sum up these steps, we summarize this via the flowchart below:

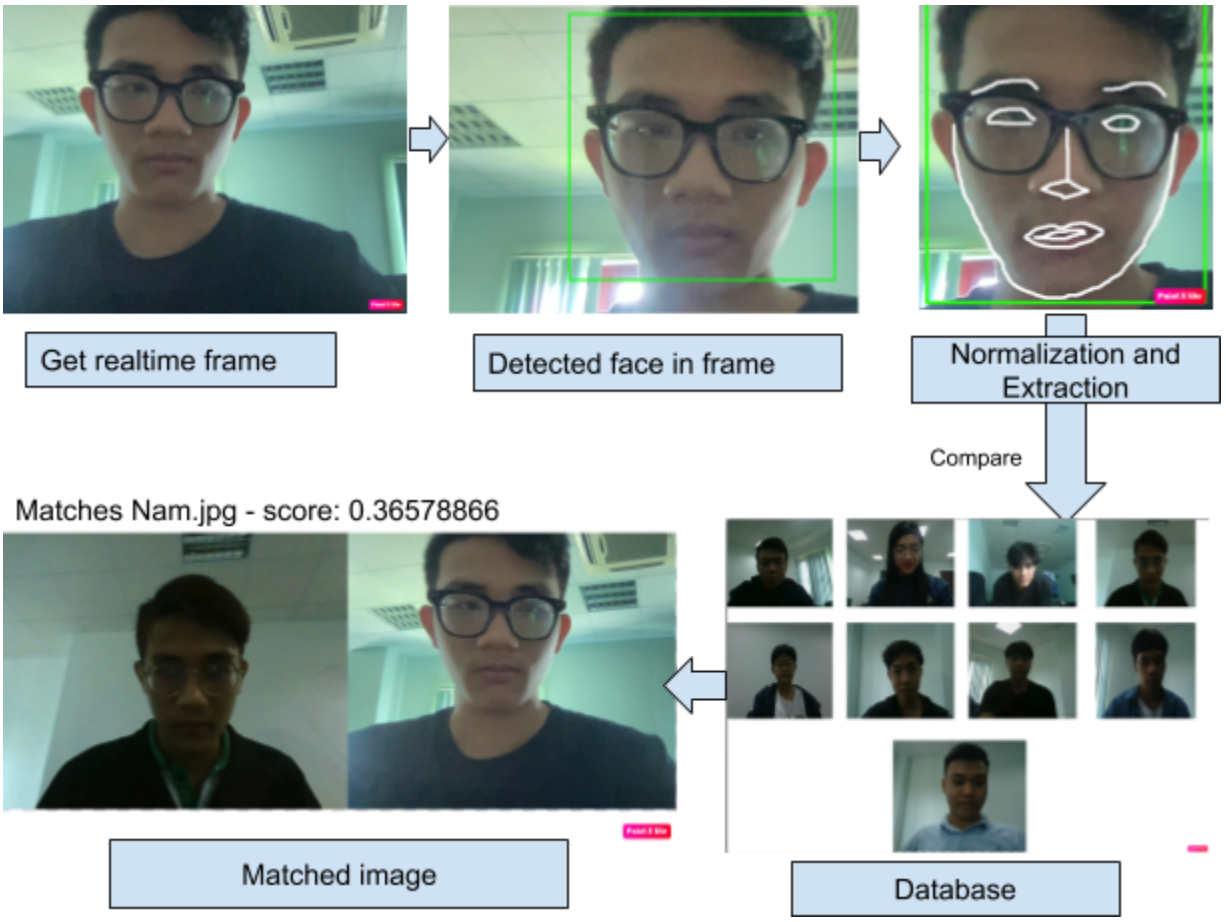



*Fig. 10. Flowchart of Face Recognition*

# VI. Result

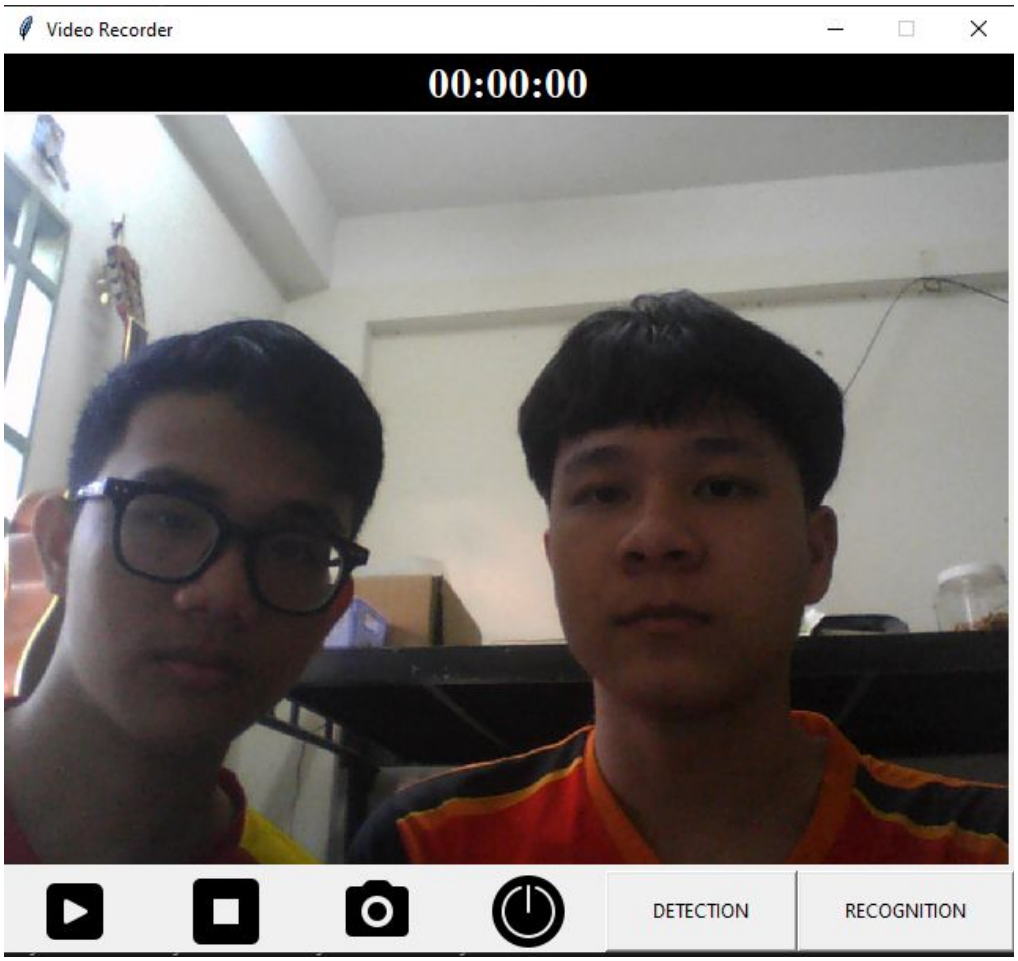



*Fig. 11. Open App*

Supporting functions for users:

1. **Take and save a photo**
2. **Recording and save video**

   User can watch the video in the unser's library
3. **Detecting face**

   The user's face is identified on the screen by a green rectangle.

Example:

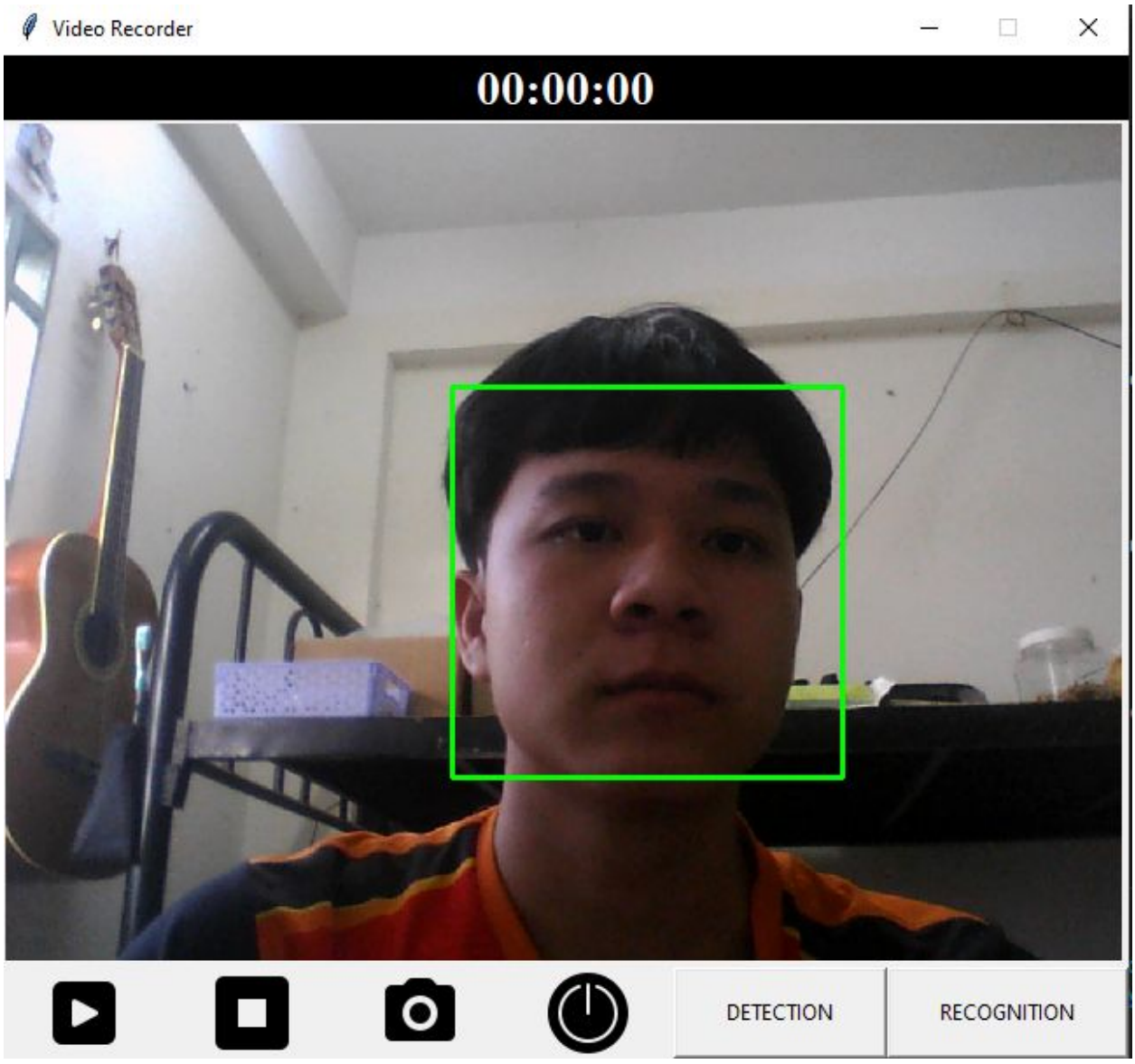



*Fig. 12. Detection*

4. **Recognition face**

The username is defined and is displayed under the terminal

Example:

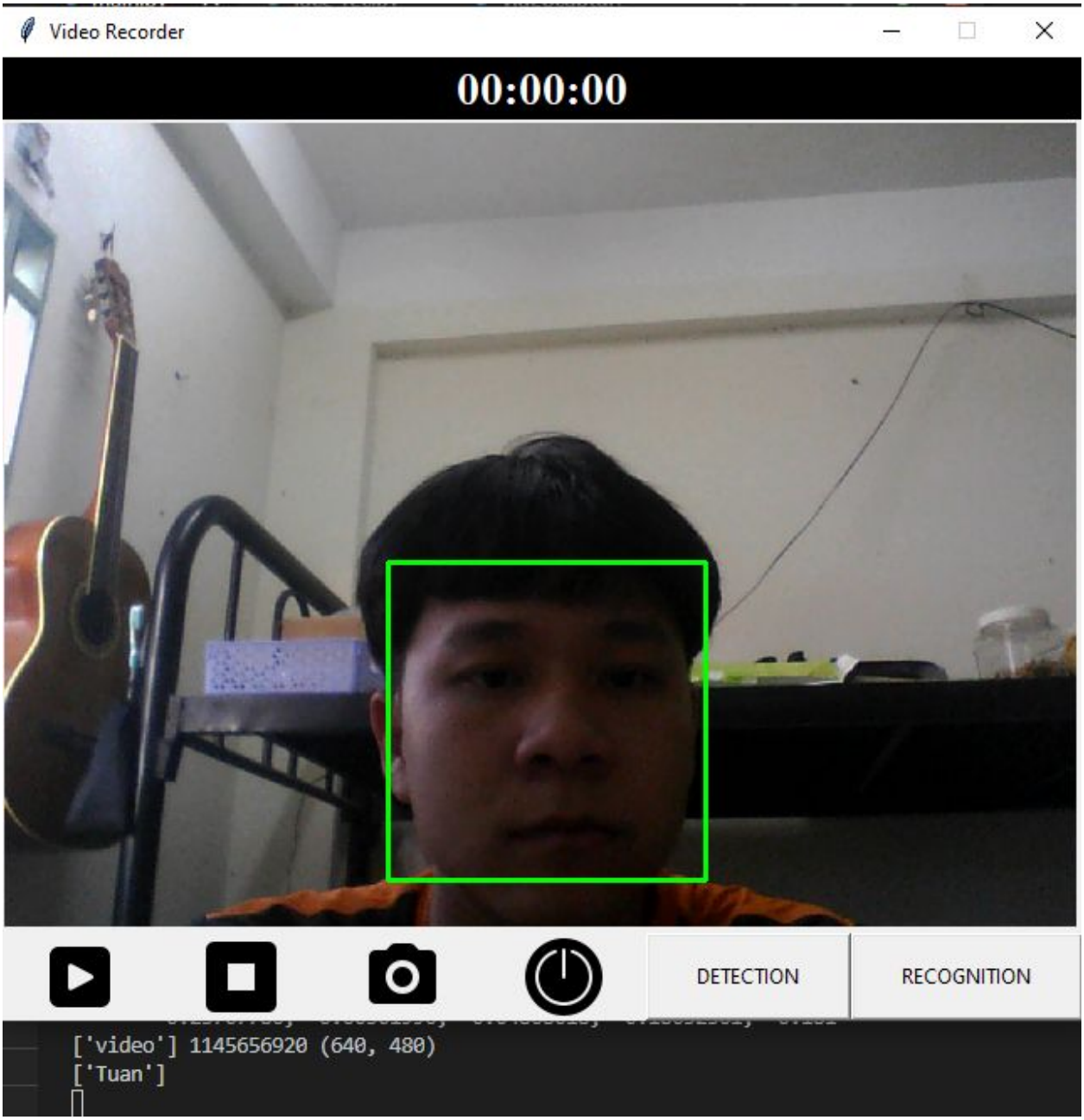



*Fig.13. Recognition*

### Limitation:

- Not being able to open the saved video on the GUI
- Computationally **complex and slow**
- **Longer appearing** result with face recognition
- Less accurate with **black faces or large scale people**
- Limitation in difficult **brightening** conditions

# VII. Conclusion

In this paper, we have concluded that the combination of Viola Jones algorithm and Discrete Cosine Transform (DCT) algorithm gives results with fast detection and high accuracy. Some of the limitations of these techniques are the high computation time when the image size is large and the resolution is high, and if high size data is used, problems arise.

The future work is to increase the accuracy of facial recognition and recognition. The Viola Jones algorithm is a fast face detection technique but it has some false positives for images with obscured faces, so future work is to try to reduce false positives.

# References:

Ageitgey Contributor, (2020). Face_recognition. Github repository. Retrieved from https://github.com/ageitgey/face_recognition

Anirudh Rao OpenCV Python Tutorial: Computer Vision With OpenCV In Python. Nov. 2020 https://www.edureka.co/blog/python-opencv-tutorial/

Black-shadows. (2018). Attendance-Using-Face-Recognition. Github repository. Retrieved from https://github.com/black-shadows/Attendance-Using-Face-Recognition

Cao, Dung. *CS111: Introduction to Computer Science: Session: Tkinter-Gui in python. Oct. 2020,* *http://courses.ttu.edu.vn/pluginfile.php/2311/mod_resource/content/1/07.Tkinter.pdf*

Cao, Dung. *CS111: Introduction to Computer Science; Session 4: Procedural Decomposition*. Oct. 2020, http://courses.ttu.edu.vn/pluginfile.php/2079/mod_resource/content/1/04.Function.pdf

Cao, Dung. *CS111: Introduction to Computer Science; Session 13: numpy library*. Nov. 2020, http://courses.ttu.edu.vn/pluginfile.php/2953/mod_resource/content/1/13.Numpy.pd

Cao, Dung. *CS111: Introduction to Computer Science; Session 10: Object-Oriented Programming Introduction*. Nov. 2020, http://courses.ttu.edu.vn/pluginfile.php/2759/mod_resource/content/1/08.OPP.pdf

Kasinski, A., & Schmidt, A. (2010). The architecture and performance of the face and eyes detection system based on the Haar cascade classifiers. *Pattern Analysis and Applications*, *13*(2), 197-211.

Padilla, R., Costa Filho, C. F. F., & Costa, M. G. F. (2012). Evaluation of haar cascade classifiers designed for face detection. *World Academy of Science, Engineering and Technology*, *64*, 362-365.

Soo, S. (2014). Object detection using Haar-cascade Classifier. Institute of Computer Science, University of Tartu, 1-12.

Tech-with-Tim. (2019). Face recognition python tutorial. Youtube. Retrieved from https://www.youtube.com/watch?v=D5xqcGk6LEc&t=434s