Republic of the Philippines
POLYTECHNIC UNIVERSITY OF THE PHILIPPINES
BACHELOR OF SCIENCE IN COMPUTER ENGINEERING
BIÑAN CAMPUS

Emerging Technologies in Computer Engineering

(Image & Video Face Detect)

By:

Bryan Yj Actub

Submitted to:

Simon Salvador Tidon

Face Detect using Python openCV (Image & Video)

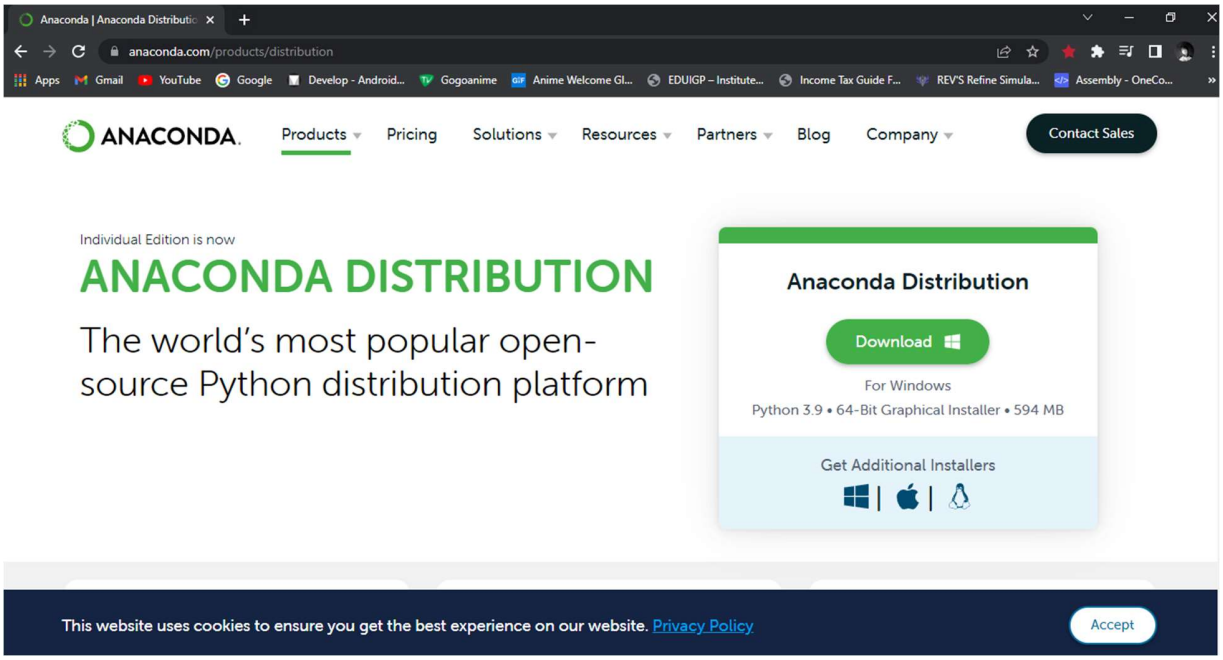

Figure 1.1

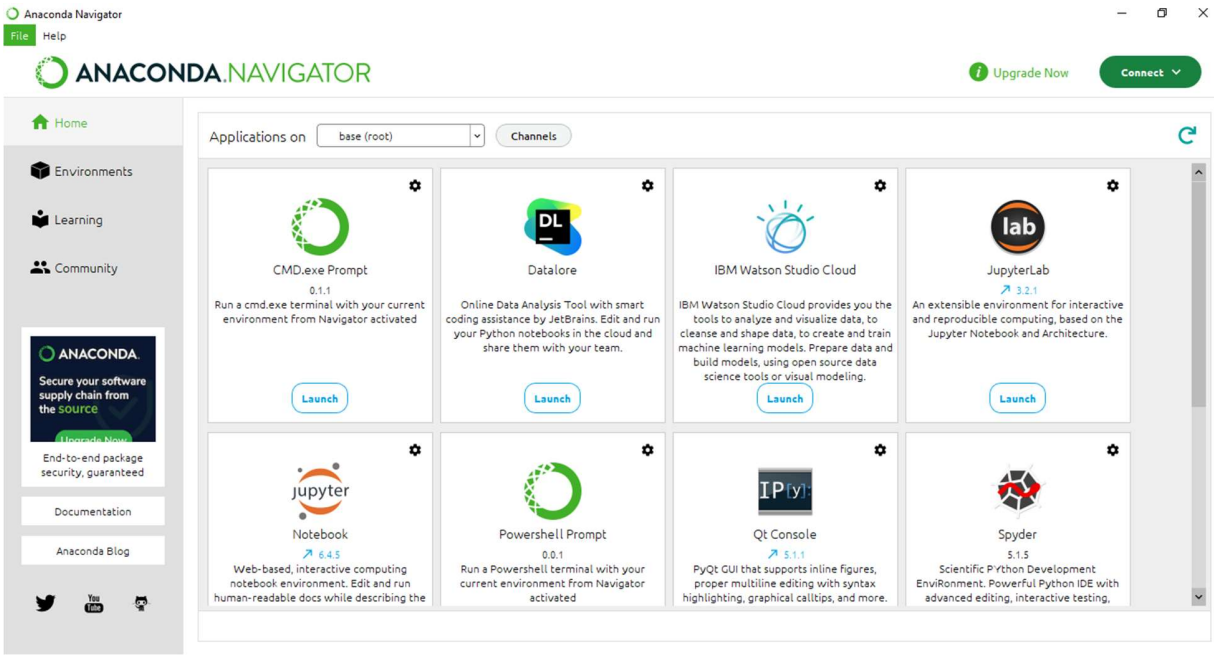

Figure 1.2

First, I install Anaconda Navigator on their website. Installing Anaconda platform will install Python, NumPy, Jupyter, Pip, and etc., that will be necessary for the project.

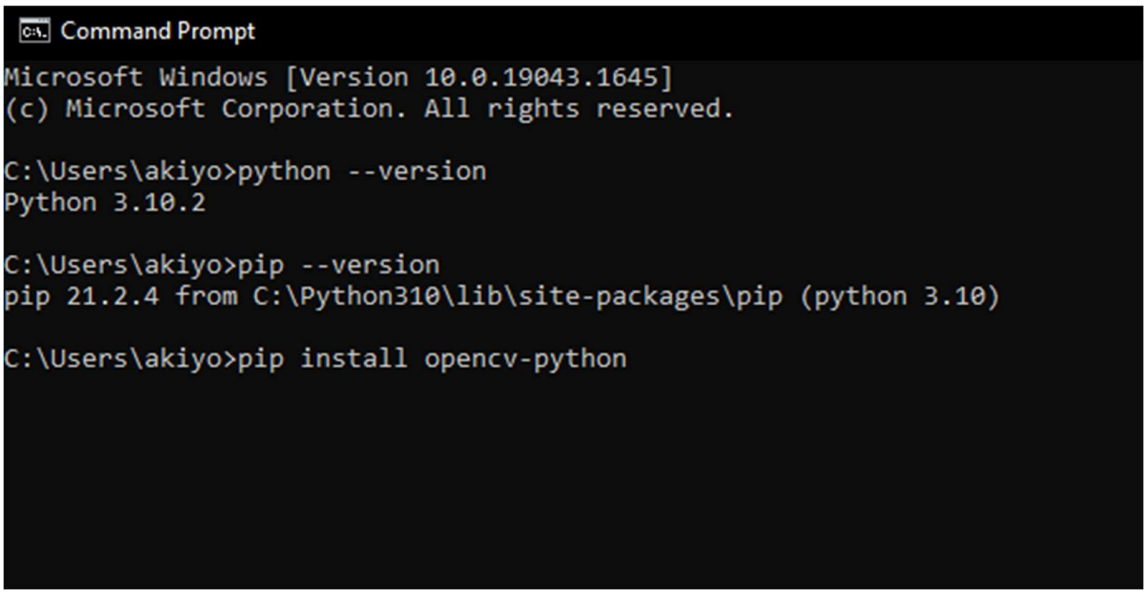

Figure 1.3

I double check if python and pip is installed by checking its version, before continuing to install OpenCV using pip “pip install opencv-python”.

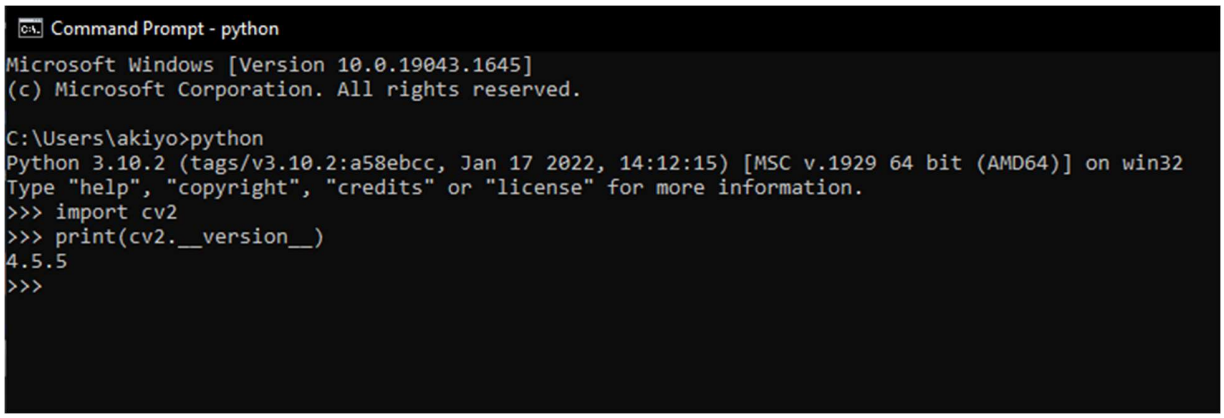

Figure 1.4

After installing OpenCV, I double check by typing import cv2 if it did not produce any error means it is installed and checked its version.

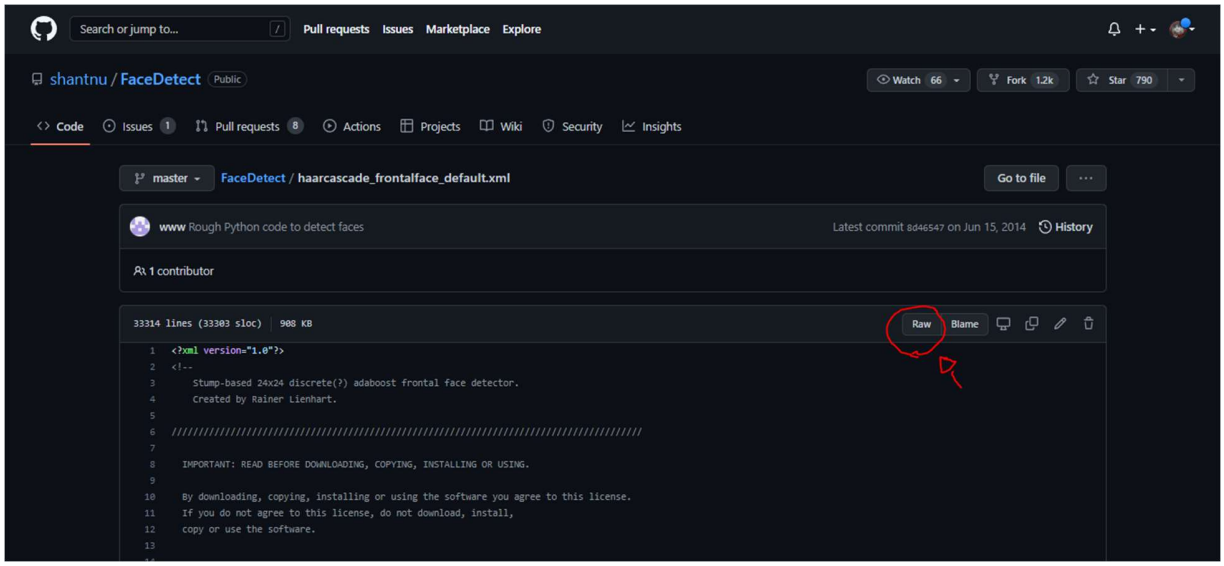

Figure 2.1

Next step is I went to Face Detect repository and download the xml file haarcascade_frontalface_default.xml, click the file and “raw” then right-click save as xml file and put it in a folder of your project.

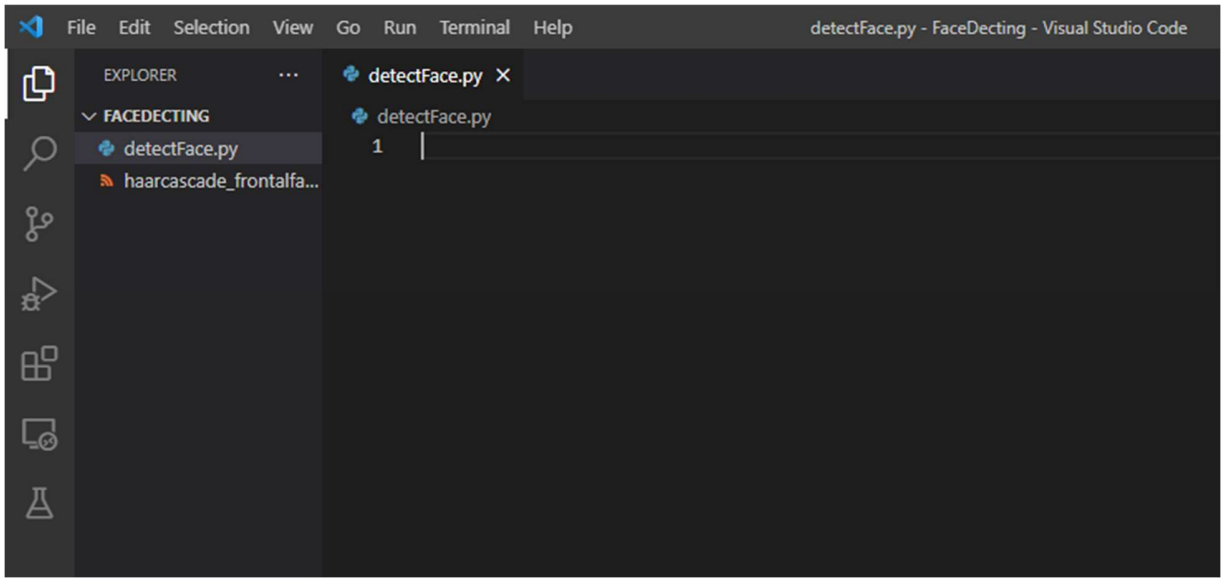

Figure 2.2

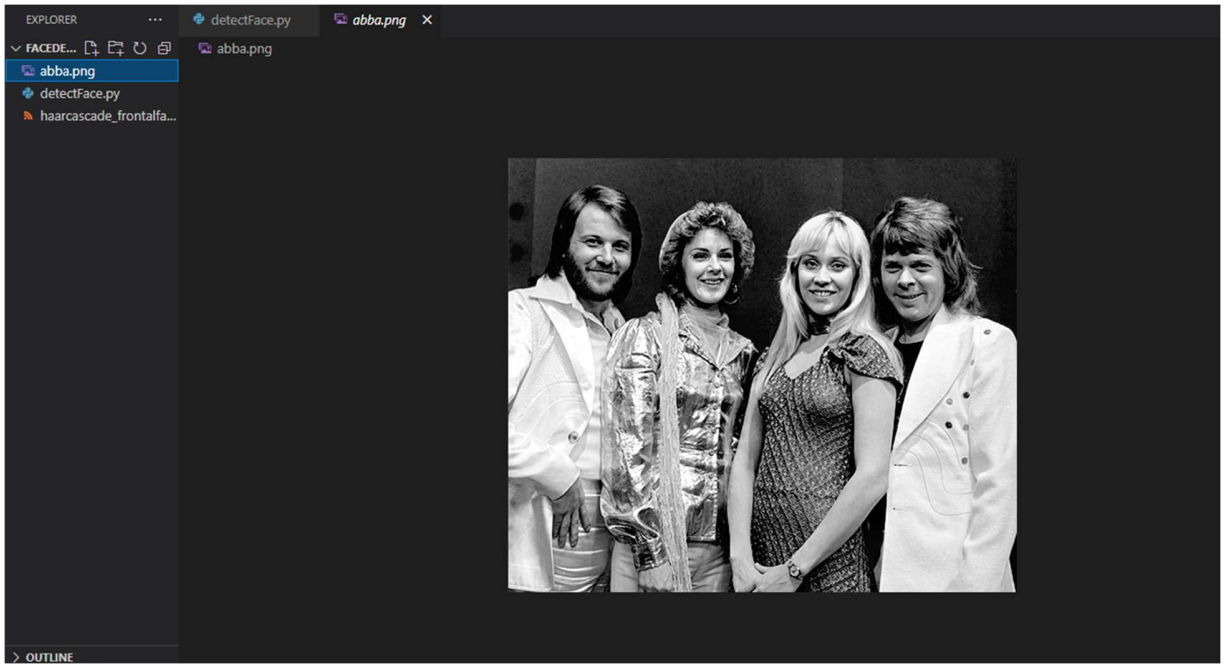

Figure 3.3

Next is to create a python file and added a picture or image to detect faces in the same directory before proceeding to code the python file.

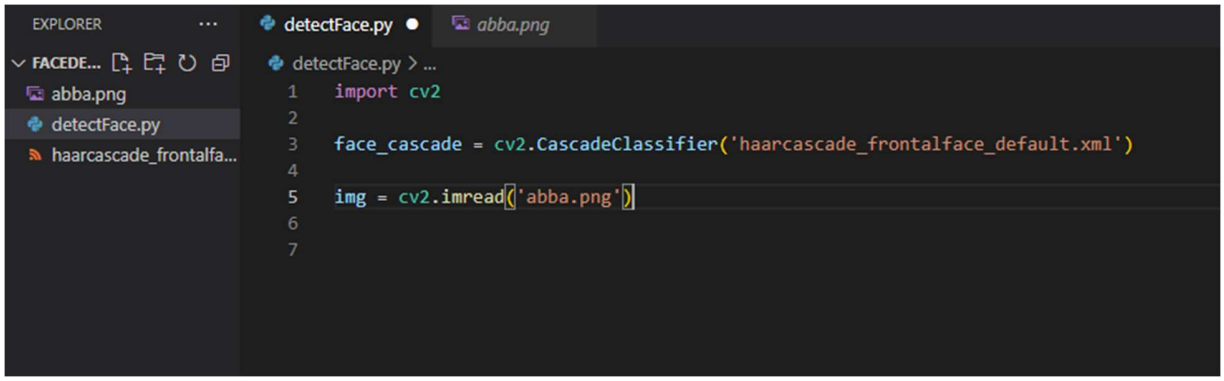

Figure 3.4

Load the classifier, the xml file that we downloaded on the Face Detect repository and create a variable to read the input image.

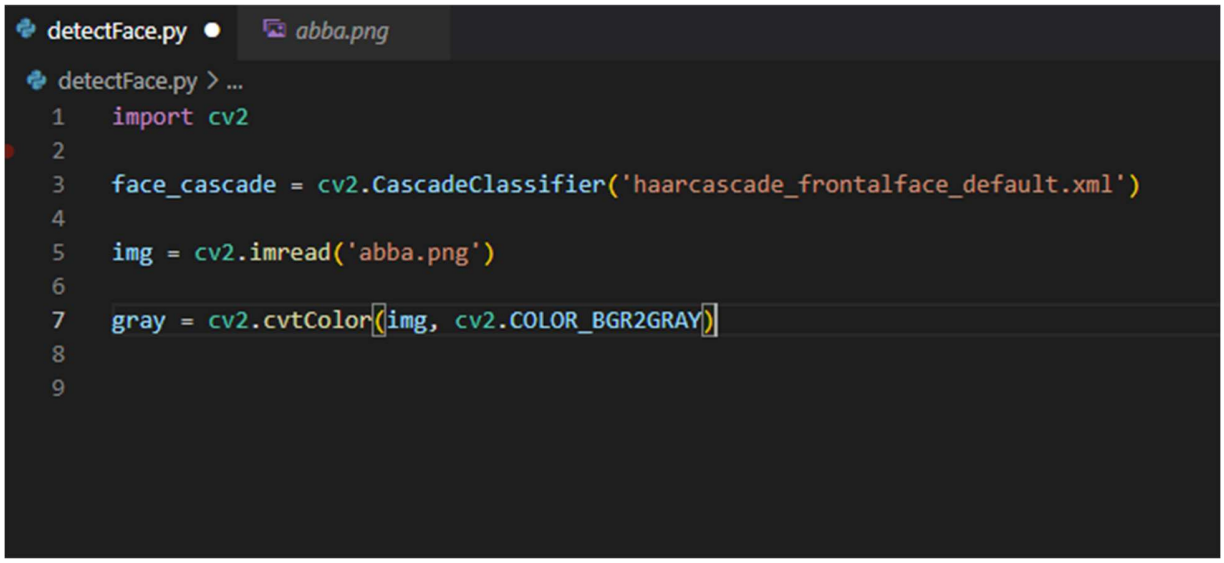

Figure 3.5

This method only works in gray scale images, we have to convert the RGB image into grayscale (line 7).

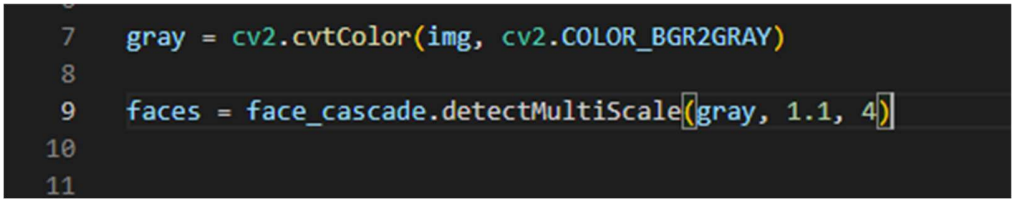

Figure 3.6

Next line is to detect faces using **detectMultiScale** function, it takes 3 arguments; the input image, scaleFactor, and minNeighbours. ScaleFactor specifies how much the image size is reduced with each scale. MinNeighbours parameter specifying how many neighbors each candidate rectangle should have to retain it. This parameter will affect the quality of the detected faces: higher value results in less detections but with higher quality.

```
9   faces = face_cascade.detectMultiScale(gray, 1.1, 4)
10
11  for (x, y, w, h) in faces:
12      cv2.rectangle(img, (x, y), (x+w, y+h), (255, 0, 0), 2)
13
14
```

Figure 3.7

Faces contains a list of coordinates for the rectangular regions where faces were found. We use these coordinates (x, y, w, h) to draw the rectangles in our image. (255, 0, 0) is the color of the outline of rectangle, in my code it is color blue because in cv2 it processes images in BGR. Blue is 255, Green and Red is 0. The “2” in the code is the thickness of the rectangle.\

```
13
14  cv2.imshow('img', img)
15  cv2.waitKey()
16
17
```

Figure 3.8

This line of code is to show the image and waitKey() is waiting for a key press to close the image. Next is to save the file and run the python file.

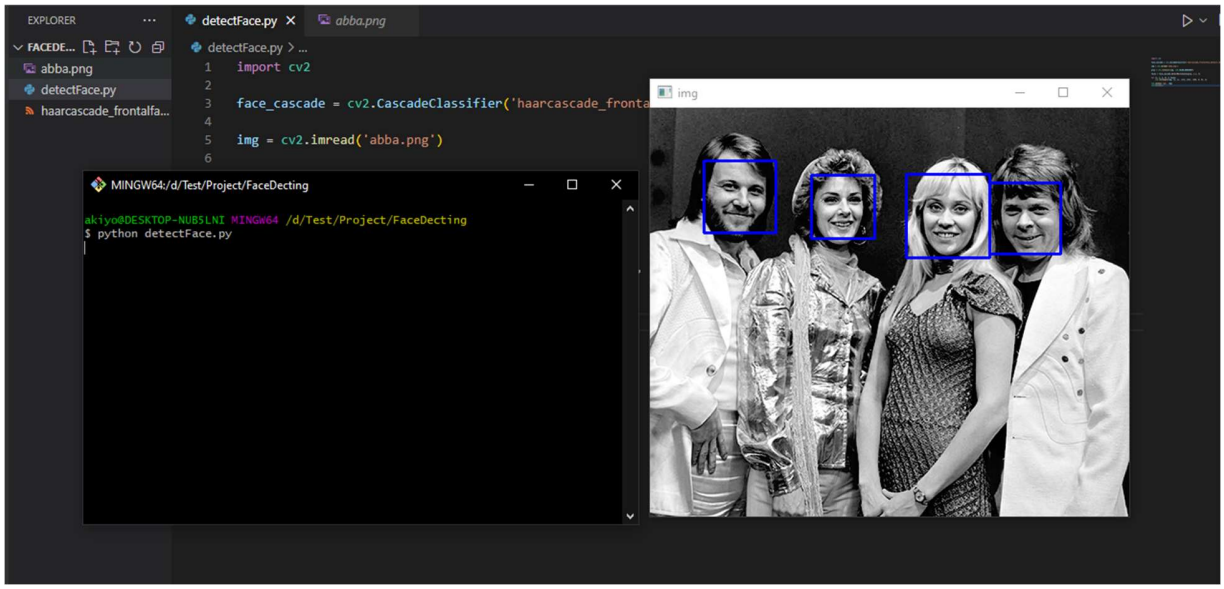

Figure 3.9

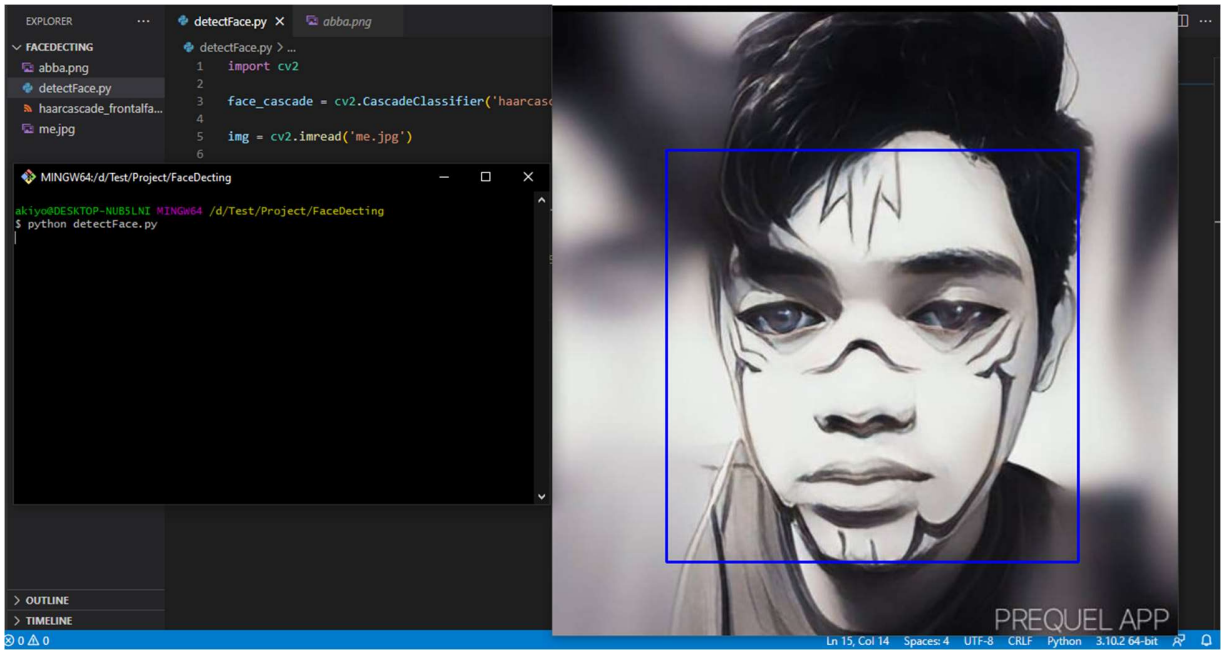

Figure 3.10

Run the python file by typing the command “python Filename.py”. Try using different images and change values in scaleFactor and minNeighbours.

Video Face Detection

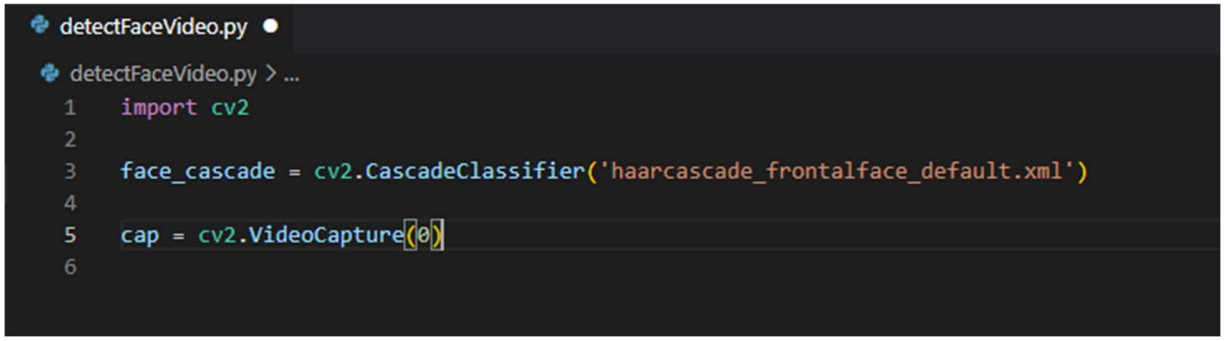

Figure 4.1

For video face detect, it is almost the same code. First is import cv2 and load the classifier (xml file). Next line is to load the video capture device if you are using multiple webcams you can change the value of 0 into 1 or 2 depending on the camera.

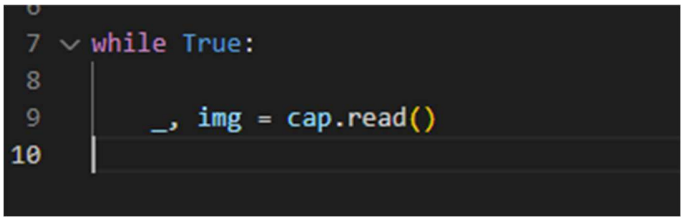

Figure 4.2

The while loop will continue running or displaying the video until a specific key in waitKey is pressed. Cap.read() this is going to return a frame or image in img variable and the "_" is a variable that will tell us if the capture works properly. For example a camera or webcam is already used for google meeting and you are trying to run this python file, the problem is the camera is already being used by other resources so the "_" will return **false** and will end the program.

```
while True:

    _, img = cap.read()

    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    faces = face_cascade.detectMultiScale(gray, 1.1, 4)

    for (x, y, w, h) in faces:
        cv2.rectangle(img, (x, y), (x+w, y+h), (255, 0, 0), 2)

    cv2.imshow('img', img)
```

Figure 4.3

Next line of codes is similar to the image detection. Converting image into grayscale, detecting faces using **detectMultiScale** function, using coordinates (x, y, w, h) to draw a rectangle (color and thickness is specified), and the display.

```
    cv2.imshow('img', img)

    k = cv2.waitKey(30) & 0xff
    if k==27:
        break

cap.release()
```

Figure 4.4

The waitKey will stop the loop if a specific key is pressed, in this case k is equals to 27, 27 is the escape key on keyboard. So, pressing the esc button will end the display of video or simply press “ctrl + c” on keyboard to terminate the current process that is running. Instead of esc key (27) you can change it into “ord(‘AnyKey’)” change the AnyKey word into a key for example “q” now by pressing “q” button will end the loop. Cap.release will release the webcam so the other resources can use the webcam when the program end. Then save and run the python code.

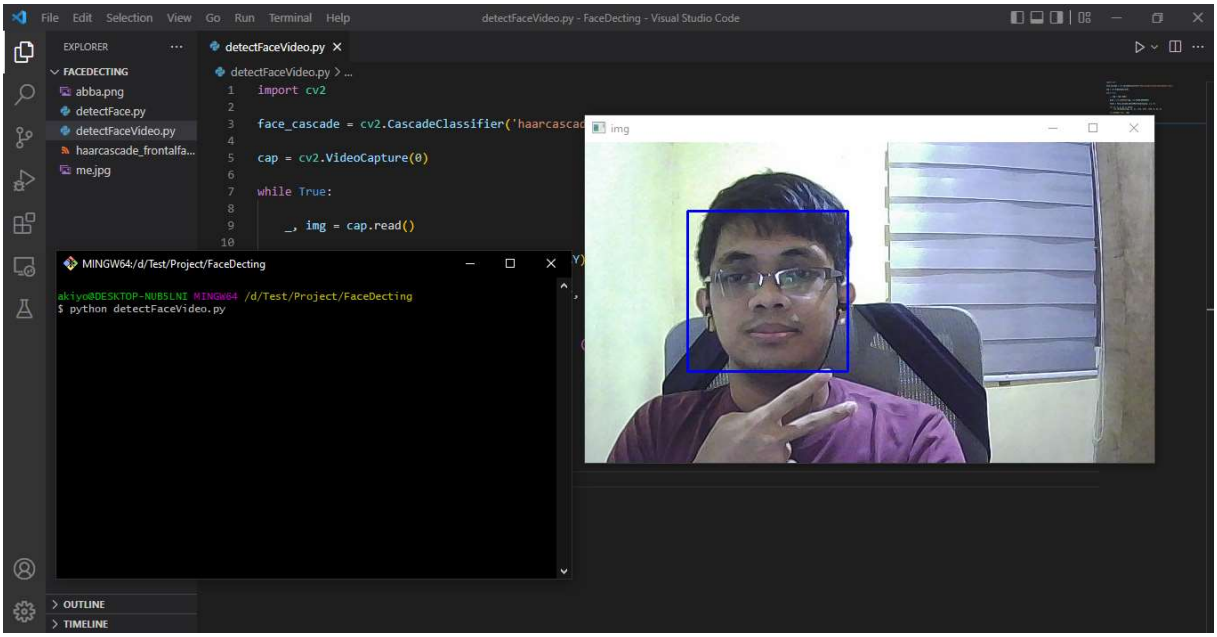

Figure 4.5

Run the python file by typing the command “python Filename.py”.

Code for image face detect;

```
import cv2

face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

img = cv2.imread('me.jpg')

gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

faces = face_cascade.detectMultiScale(gray, 1.1, 4)

for (x, y, w, h) in faces:
    cv2.rectangle(img, (x, y), (x+w, y+h), (255, 0, 0), 2)
```

```
cv2.imshow('img', img)
cv2.waitKey()
```

Code for video face detect;

```
import cv2

face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

cap = cv2.VideoCapture(0)

while True:

    _, img = cap.read()

    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    faces = face_cascade.detectMultiScale(gray, 1.1, 4)

    for (x, y, w, h) in faces:
        cv2.rectangle(img, (x, y), (x+w, y+h), (255, 0, 0), 2)

    cv2.imshow('img', img)

    k = cv2.waitKey(30) & 0xff
    if k==27:
        break

cap.release()
```