Chapter – 1 (implementation – Part 2 - Homework)

**Computer Vision**

**OpenCV: Happiness Detector**

Happiness Detector

Implementation

**1.8 Happiness Detect – Problem Description**

* Description: We want to build a computer vision application that can detect smile from our face. It can be very useful to understand customer's reactions when for example they're watching a movie.
* Some *companies* indeed use some *computer vision tools* to recognize *customers expressions* when they're watching a movie to understand what makes audience smile what makes them sad.
* This kind of tool helps to understand the audience.
* Also it can be applied to a *recommender system* to recommend some new movies which will boost audience's emotions.
* Project Phases: We devide this prject into several phases:

1. RESEARCH Phase: When you are a **computer vision scientist** or even a **machine-learning scientist** or an **AI-engineer**, there is always the research phase to look for some solutions.

* Our research phase consists of finding the **"Haar cascade for smile"** that will help us build this smile detector.

1. IMPLEMENTATION phase: In this phase we'll of course use the previous code to be more efficient.
2. TESTING Phase: We will watch the result in this 3rd phase to notice that is it optimal or not.
3. TUNING phase: In this Phase Four we will do some "PARAMETER TUNING" to improve the app, to make it a much more accurate smile detector.

**1.9 Happiness Detect – Implementation**

1. RESEARCH Phase: We're going to find the right cascade to detect the smile.

* We'll find a place where you have all the Haar cascades in case we want to build something else.
* Search: Google "OpenCV Haar cascade" .
* We go to the **GitHub repository of OpenCV**. Following is the link of the repo:

<https://github.com/opencv/opencv/tree/master/data/haarcascades>

In this link we will find all the different Haar cascades that are allowed to detect different-features.

Most of them are face features not only human face but also some cat-face features. There are also cascades to detect Russian plate number.

* However, we are interested in "haarcascade\_smile.xml". We'll add this to our previous code and we will need to change some parameters in the implementation to make it work better.
* Extracting: "Right-Click and Save Link as" won't work. You have to open these xml files in the browser, copy the content and save the content using a text editor. Give the ".xml" extension.

1. IMPLEMENTATION phase: Open spyder in virtual environment, use "conda env list" then select the environment "conda activate env\_name", type "spyder" to lunch it.

* Our project folder contains the four files. These are:

haarcascade\_eye.xml

haarcascade\_frontalface\_default.xml

haarcascade\_smile.xml

smile\_detection.py

* Notice we still need **haarcascade\_eye.xml** and **haarcascade\_frontalface\_default.xml** because we will still be detecting the eye and the face.

Use the previous codes:

* Changes: Here we load the Cascades for smile along with other two cascades for face and eye.

smile\_cascade = **cv2.CascadeClassifier**('haarcascade\_smile.xml')

* Inside the **detect()** function we need to add the code that will detect a smile in the face (or multiple smells if there are multiple faces) and draw the rectangles around the smile.
* Note that, we draw the rectangle in the referential of face (as we did for the eye detection). Inside the for-loop for the face we add following code:

        # *We apply the detectMultiScale method to locate smile in the face.*

        smile = **smile\_cascade.detectMultiScale**(roi\_gray, 1.1, 3)

        # *For each detected smile:*

**for** (sx, sy, sw, sh) **in** smile:

            # *We paint a rectangle around the smile, but inside the referential of the face.*

**cv2.rectangle**(roi\_color, (sx, sy), (sx+sw, sy+sh), (0, 0, 255), 2)

* roi\_gray is the region of the interest of the face, scaling factor 1.1 and min no. of neighbor 3. We'll tune these parameters later.
* Inside the for-loop: we rename the coordinates and change the rectangle color to (0, 0, 255) i.e blue.
* We will have three different colors to detect the face, eye and smile in blue, green and red respectively.

**All code before parameter-tuning**

# *---------------    Smile Recognition using OpenCV and HaarCascade    ---------------*

**import** cv2

#*----------    Loadding the cascades    ----------*

# *We load the cascade for the face, eyes & smile*

face\_cascade = **cv2.CascadeClassifier**('haarcascade\_frontalface\_default.xml')

eye\_cascade = **cv2.CascadeClassifier**('haarcascade\_eye.xml')

smile\_cascade = **cv2.CascadeClassifier**('haarcascade\_smile.xml')

#*----------    Defining a function that will do the detections    ----------*

# *We create a function that takes as input the image in black and white (gray) and the original image (frame),*

    # *and that will return the same image with the detector rectangles.*

**def** **detect**(gray, frame):

    # *We apply the detectMultiScale method from the face\_cascade to locate one or several faces in the image.*

    faces = **face\_cascade.detectMultiScale**(gray, 1.3, 5)

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

**cv2.rectangle**(frame, (x, y), (x+w, y+h), (255, 0, 0), 2) # *rectangle around the face.*

        roi\_gray = gray[y:y+h, x:x+w]       # *region of interest in the black and white image.*

        roi\_color = frame[y:y+h, x:x+w]     # *region of interest in the colored image.*

        # *We apply the detectMultiScale method to locate one or several eyes in the face.*

        eyes = **eye\_cascade.detectMultiScale**(roi\_gray, 1.1, 3)

        # *For each detected eye:*

**for** (ex, ey, ew, eh) **in** eyes:

            # *We paint a rectangle around the eyes, but inside the referential of the face.*

**cv2.rectangle**(roi\_color,(ex, ey),(ex+ew, ey+eh), (0, 255, 0), 2)

        # *We apply the detectMultiScale method to locate smile in the face.*

        smile = **smile\_cascade.detectMultiScale**(roi\_gray, 1.1, 3)

        # *For each detected smile:*

**for** (sx, sy, sw, sh) **in** smile:

            # *We paint a rectangle around the smile, but inside the referential of the face.*

**cv2.rectangle**(roi\_color, (sx, sy), (sx+sw, sy+sh), (0, 0, 255), 2)

**return** frame # *We return the image with the detector rectangles.*

#*----------    Capturing from WEBCAM    ----------*

video\_capture = **cv2.VideoCapture**(0) # *turn the webcam on.*

# *0: internal webcam,*

# *1: external webcam*

**while** **True**: # *We repeat infinitely (until break):*

    \_, frame = **video\_capture.read**() # *We get the last frame.*

    gray = **cv2.cvtColor**(frame, cv2.COLOR\_BGR2GRAY) # *We do some colour transformations.*

    canvas = **detect**(gray, frame) # *We get the output of our detect function.*

**cv2.imshow**('Video', canvas) # *We display the outputs.*

**if** **cv2.waitKey**(1) & 0xFF **==** **ord**('q'): # *If we type on the keyboard:*

**break** # *We stop the loop.*

**video\_capture.release**() # *We turn the webcam off.*

**cv2.destroyAllWindows**() # *We destroy all the windows inside which the images were displayed.*

|  |  |
| --- | --- |
| 1. TESTING Phase: At the first try we get some **false-detection**, because we need to do some **parameter-tuning**.  * We don't have a good smile detector here so we have to do parameter-tuning. * In the end we should have only one red rectangle around our smile  1. Tuning Phase: We have to tune the parameters in the **detectMultiScale()** method. 2. roi\_gray: We're not going to replace **roi\_gray** by **gray** because then we will see some red rectangles all around our face. We want to make the detection in the referential of the face for the simple reason that *"a smile is in the face"* that's why we're keeping **roi\_gray**. |  |

1. scaling factor: We're going to choose a **larger scaling factor** and we're going to increase it to **1.7**. However, it doesn't do the most improvement but still it will help a little.
2. number of neighbors (smile): This is the parameter that will definitely make a difference. We *increase* the *number of neighbors* because in some way that the detection has to be much more thorough.

* If we have a low number of minimal neighbors, anything that looks approximately like a smile will be detected as a smile and that will be not good.
* And that's why we obtained many rectangles (red-color for smile) in the previous detection.
* The right number is obtained with experimentation. And this number is 22.
* We increased the number of minimum neighbors from ***3*** to ***22***.

1. Number of neighbors (eyes): We're going to increase the minimum number of neighbors of the eyes and we choose **16** (or try **22**).

>> activate py354

>> spyder

**Parameter Tuned Code**

# *---------------    Smile Recognition using OpenCV and HaarCascade    ---------------*

**import** cv2

#*----------    Loadding the cascades    ----------*

# *We load the cascade for the face, eyes & smile*

face\_cascade = **cv2.CascadeClassifier**('haarcascade\_frontalface\_default.xml')

eye\_cascade = **cv2.CascadeClassifier**('haarcascade\_eye.xml')

smile\_cascade = **cv2.CascadeClassifier**('haarcascade\_smile.xml')

#*----------    Defining a function that will do the detections    ----------*

# *We create a function that takes as input the image in black and white (gray) and the original image (frame),*

    # *and that will return the same image with the detector rectangles.*

**def** **detect**(gray, frame):

    # *We apply the detectMultiScale method from the face\_cascade to locate one or several faces in the image.*

    faces = **face\_cascade.detectMultiScale**(gray, 1.3, 5)

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

**cv2.rectangle**(frame, (x, y), (x+w, y+h), (255, 0, 0), 2) # *rectangle around the face.*

        roi\_gray = gray[y:y+h, x:x+w]       # *region of interest in the black and white image.*

        roi\_color = frame[y:y+h, x:x+w]     # *region of interest in the colored image.*

        # *We apply the detectMultiScale method to locate one or several eyes in the face.*

        eyes = **eye\_cascade.detectMultiScale**(roi\_gray, 1.1, 16)

        # *For each detected eye:*

**for** (ex, ey, ew, eh) **in** eyes:

            # *We paint a rectangle around the eyes, but inside the referential of the face.*

**cv2.rectangle**(roi\_color,(ex, ey),(ex+ew, ey+eh), (0, 255, 0), 2)

        # *We apply the detectMultiScale method to locate smile in the face.*

        smile = **smile\_cascade.detectMultiScale**(roi\_gray, 1.7, 16)

        # *For each detected smile:*

**for** (sx, sy, sw, sh) **in** smile:

            # *We paint a rectangle around the smile, but inside the referential of the face.*

**cv2.rectangle**(roi\_color, (sx, sy), (sx+sw, sy+sh), (0, 0, 255), 2)

**return** frame # *We return the image with the detector rectangles.*

#*----------    Capturing from WEBCAM    ----------*

video\_capture = **cv2.VideoCapture**(0) # *turn the webcam on.*

# *0: internal webcam,*

# *1: external webcam*

**while** **True**: # *We repeat infinitely (until break):*

    \_, frame = **video\_capture.read**() # *We get the last frame.*

    gray = **cv2.cvtColor**(frame, cv2.COLOR\_BGR2GRAY) # *We do some colour transformations.*

    canvas = **detect**(gray, frame) # *We get the output of our detect function.*

**cv2.imshow**('Video', canvas) # *We display the outputs.*

**if** **cv2.waitKey**(1) & 0xFF **==** **ord**('q'): # *If we type on the keyboard:*

**break** # *We stop the loop.*

**video\_capture.release**() # *We turn the webcam off.*

**cv2.destroyAllWindows**() # *We destroy all the windows inside which the images were displayed.*