Chapter – 1 (implementation – Part 2 - implementation)

**Computer Vision**

**OpenCV: face detection**

Intro

Installation

Implementation

**1.2 Implementation: Step 1 (import libraries and cascade files)**

* In the intuition part of this chapter, we now know how the Cascades work.
* We're going to apply cascade to do some face recognition,
* We're going to do it through the *web-cam* and we're going to try to predict our *face*, *eyes* and *mouth*, *smile* etc.
* We'll also use ***OpenCV*** to do this face recognition (We'll do much better models with object detection, it's much more powerful than **open-cv**.).
* Cascade-XML files: Make sure these two XML files: "haarcascade\_eye.xml" & "haarcascade\_frontalface\_default.xml" are present in the working directory.
* We will be using the ***Cascades*** from these files to ***execute*** our ***code***.
* "haarcascade\_eye.xml" is the cascade for the eye.
* "haarcascade\_frontalface\_default.xml" is to detect the frontal face.
* Libraries: The only library that we need here is **open-cv**. and that is silly to say Make sure you
* Note that, we're not doing deep learning here, so we *don't need* *import* more *libraries*.
* From the intuition lectures, we know that **Cascades** are not neural networks. They're actually a deep series of FILTERS not a series of layers (the *cascade* is a *series of filters* that will apply one after the other to detect the face). So ***open-cv-cascades*** are not based on any neural networks.
* Hence we don't need to import any neural network libraries like the Torch or TensorFlow.
* We only need ***cascades*** from the ***open-cv***.
* Loading the Cascades: Now we load cascade, one for eye and one for the face.
* To do this we're going to create two objects from a class called ***CascadeClassifier***. This class is from the cv2 module from the open-cv library.
* This class will take the "XML files that contains the cascade" as arguments.
* The first object for the frontal face cascade and
* The second object for the eye cascade.

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

**import** cv2

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

# *We load the cascade for the face.*

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

# *We load the cascade for the eyes.*

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

Next, we will implement a function that will do the detection by using those cascades to detect the face and the eye.

We'll use a little trick for computational efficiency: by doing the face detection on the *global referential* but eye defecation with the *referential of the face*. It will save us some computation time.

**1.3 Implementation: Step 2 (face detect function)**

Now we'll define a function that will do the detections.

* First it gets the ***coordinates*** of the ***rectangle*** that will detect the ***face*** and the ***eyes***.
* Then we will have a **for-loop** that will iterate through the different faces there are detected in the video.
* For each of these faces we will draw a rectangle that indicates a face.
* Also inside each of these faces we will detect some eyes.

#*----------    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)

* Arguments: This function is going to be applied on single images. It's going to return the same image with the rectangle detecting the faces and the eyes.
* So it's going to take an image as inputs. It's going to be the different images of the video coming from the webcam. But this function works on single images one by one.
* That's why we can't put the video directly we have to applay it on images.

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

* Two arguments: From the intuition part we know that a cascade works on Black and White image.
* So at first we convert the image into black and white version. And then at the end this function will return the original image with the rectangles.
* **grey:** which is for the black and white image
* **frame:** indicates the original frame (the original image) which we will call frame
* The coordinates of the rectangle that will detect the face: We're going to get some touples of four elements **(x, y, w, h)**.
* **x** and **y** are the coordinates of the upper left corner.
* **w** = width of the rectangle and
* **h** = height of the rectangle.
* Since we're going to get several touples, we're going to put all these in a variable that we're going to call ***faces***.
* We are going to get these touples using the ***CascadeClassifier.detectMultiScale*** method.
* Since we are working with faces, we'll take ***face\_cascade*** which is a object of ***CascadeClassifier***.
* This method will get us to the coordinates of the upper left corner and the width and height of the rectangles detecting the faces.
* Arguments:
* **gray:** To convert the image in Black-white.
* **scale factor:** which tells by how much the size of the image is going to be reduced. Or equivalently by how much the size of the filter will be increased.

We are going to take a **scaling factor** of **1.3** which means that the size of the image will be reduced **1.3 times**.

* **Minimum number of neighbors:** From intuition part, we saw that in order for a zone of pixels to be accepted, we'll need to have at least a certain number of neighbor zones that are also accepted.

That certain number is exactly the minimum number that we're going to input right now and that is going to be 5.

That means that in order for a zone of pixels to be accepted one at least five neighbor zones must also be accepted.

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

* Why 1.3 and five?
* Well the reason is that we get some really good results with these numbers.
* We can always experiment (tune) them. After some tuning it turns out 1.3 and 5 is a good combo to detect some faces with the web-cam.

Next, we're going to start a for-loop to iterate through the different faces. We'll have several touples and for each of these faces we will draw the rectangle and inside these rectangles will detect some eyes.

**1.4 Implementation: Step 3 (rectangle for face)**

Now we're going to start a for-loop that will iterate through faces we're going to draw a rectangle and we will detect some eyes.

* Draw the rectangle: Using OpenCV's ***rectangle*** function, we can do this very easily.

**for** (x, y, w, h) **in** faces: # *For each detected face:*

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

Parameters:

* Image: frame
* Upper left corner: (x, y)
* Lower rignt corner: (x+w, y+h)
* Color of the rectangle: (255, 0, 0)
* Rectangle thickness: 2

Now we will get some nice rectangles for the faces.

We're going to do the same for the eyes, but to save some computation time, we're going to do that in the referential of the face.

**1.5 Implementation: Step 4 (rectangles for eyes)**

Now we'll detect the eyes in the reference of the face, i.e in the reference of the detected-rectangles for the face.

* Region of interest: To detect the eyes inside the rectangles we'll select two regions of interest.
* We need one region of interest for the black-white image in which the cascade is going to be applied to detect the eyes.
* Other region of interest is for the original colored image to draw the rectangles in the original color image.

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

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

Where

**(x, y)** is Coordinate of *upper left corner*,

**(x+w, y+h)** is Coordinate of *lower rignt corner*,

We're taking the range from ***y*** to ***y+h*** and ***x*** to ***x+w***.

* Detect the eyes: We're going to detect them in the region of interest of the gray image, because the Cascades are applied in black-white images.
* We're going to do exactly the same as what we did for the faces.
* We're going to take our **eye\_cascade** object of the **CascadeClassifier** class and we're going to apply **detectMultiScale** method to detect the eyes.
* That is we're going to get the coordinates of the upper left corner of the rectangle detecting the eyes and the width and the height of these rectangles.

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

* Notice, we applied the region of interest **roi\_gray**, instead of whole image (whole image is represented by " **gray** ").
* We reduced the scale-factor to **1.1**.
* We also reduced the minimum-no. of neighbors to **3**.
* These numbers depends on experimentation, and we can tune these values to get more accurate results in the future. For now, these values are the optimal and will give us some good results.

Next, we'll start a new (nested) for loop in which we will draw the rectangles around the eyes.

**1.6 Implementation: Step 5 (rectangle for eyes and return)**

Above we detected our eyes, we gathered the coordinates of the upper left corner, the width and the height of the rectangles detecting the eyes. Now we need to draw these rectangles (as we did for the face).

* We're going to start a new for-loop to do that inside the first for-loop because we are drawing these rectangles in the referential of the face.

**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)

* **roi\_color**: colored region of the image (region of interest)
* **(0, 255, 0)**: We choose different color for the eyes-rectangle
* Finally we want to return the original frame with the rectangles detecting the face and the eyes.
* So we need to return the "**frame**" since " roi\_color " is just a subzone of the "frame".
* It will return the *original image* with the *rectangles* detecting the *faces* and the *eyes*.

**return** frame

* Also our function is over. Following is all code (done so far) at once:

# *Face Recognition*

# *Importing the libraries*

**import** cv2

# *Loading the cascades*

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

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

# *Defining a function that will do the detections*

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

    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)

        roi\_gray = gray[y:y+h, x:x+w]

        roi\_color = frame[y:y+h, x:x+w]

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

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

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

**return** frame

In next step we'll turn on the **web-cam** and we apply this newly-defined **detect()** function on the *streaming images* coming from the *webcam*.

It'll be applied on each image one by one but at a very high rate and therefore in real time we'll be able to see some *rectangles* *surrounding* our *face* and our *eyes* in the video coming from the webcam.

**1.7 Implementation: Step 6**

In this step we're going to do the face recognition with the web cam by applying the **detect()** function that we made in the previous step. We'll apply **detect()** on each of the images coming from the webcam.

* First, we need the last frame coming from the webcam because on this *last frame* we will apply the ***detect*** function.
* To get this last frame we're going to create an object of another class called **VideoCapture** from open-cv's **cv2**.

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

# *0: internal webcam,*

# *1: external webcam*

* This object will contain several entities/information and one of them will be the last frame coming from the webcam.
* If it's the external webcam that you plugged into your computer use **1** as argument.
* For the internal webcam put **0** as argument.
* We now start an **infinite while loop** to apply to detect function on all the images coming from the webcam through this **video\_capture** object to detect the face and the eyes.
* We will break this loop by using a break trick: we type on **q** on the keyboard to *turn off the webcam*.
* Inside this *while-loop*, to get the last frame we'll use **read()** method from **video\_capture** object, it returns two elements. The second element is the last frame coming from the webcam.
* Since we're only interested in the second element we'll use *underscore to skip first element*.

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

* Since the **detect()** method takes the frame (color image) and gray (Black-white image) as input we need to convert the **frame** into grayscale using **cvtColor()** method from **cv2**.

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

* arguments:

1. **frame:** we apply **cvtColor()** on **frame** (color image got from **read**()).
2. **cv2.COLOR\_BGR2GRAY:** It takes an average on blue, green and red channel of the color image in order to get the right contrast of black and white.

**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.*

It's just a trick to obtain the right shades of gray.

* Now we apply **detect()** to get the original image with the rectangles detecting the face and the eyes.

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

**frame**: The last frame captured by the webcam

**gray**: the black and white version of this frame.

* Next, we need to display all the successive outputs (successive processed images) in an animated way.

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

* To do this we're going to use **imshow()** method from our cv2 module. That will display all these processed images in an animated way in a window.
* It takes two arguments:

**'Video'**: video animation creates

**canvas:** Is the output of from **detect()** function. COLOR images that coming from the webcam but with the detector rectangles.

* Stop the webcam: To Stop the webcam and face detection process we press "**q**" on the keyboard. It'll break the **while loop**.

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

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

* Note that, it's not very important to understand the techniques and syntax of this stopping mechanism.
* It just means that if we press "**q**" on the keyboard, this will break the process of facial recognition.
* Turn off the webcam and close the window: We use **release()** method on **video\_capture** object to turn off the webcam and to destroy the window we use **destroyAllWindows()** on our **cv2** module.

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

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

**All code at once**

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

**import** cv2

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

# *We load the cascade for the face.*

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

# *We load the cascade for the eyes.*

eye\_cascade = **cv2.CascadeClassifier**('haarcascade\_eye.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: # *For each detected face:*

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

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

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

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

        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)

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

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

video\_capture = **cv2.VideoCapture**(1) # *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.*

No comment version:

# *-----------------    no comment version   -----------------*

# *Face Recognition*

# *Importing the libraries*

**import** cv2

# *Loading the cascades*

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

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

# *Defining a function that will do the detections*

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

    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)

        roi\_gray = gray[y:y+h, x:x+w]

        roi\_color = frame[y:y+h, x:x+w]

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

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

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

**return** frame

# *Doing some Face Recognition with the webcam*

video\_capture = **cv2.VideoCapture**(0)

**while** **True**:

    \_, frame = **video\_capture.read**()

    gray = **cv2.cvtColor**(frame, cv2.COLOR\_BGR2GRAY)

    canvas = **detect**(gray, frame)

**cv2.imshow**('Video', canvas)

**if** **cv2.waitKey**(1) & 0xFF **==** **ord**('q'):

**break**

**video\_capture.release**()

**cv2.destroyAllWindows**()