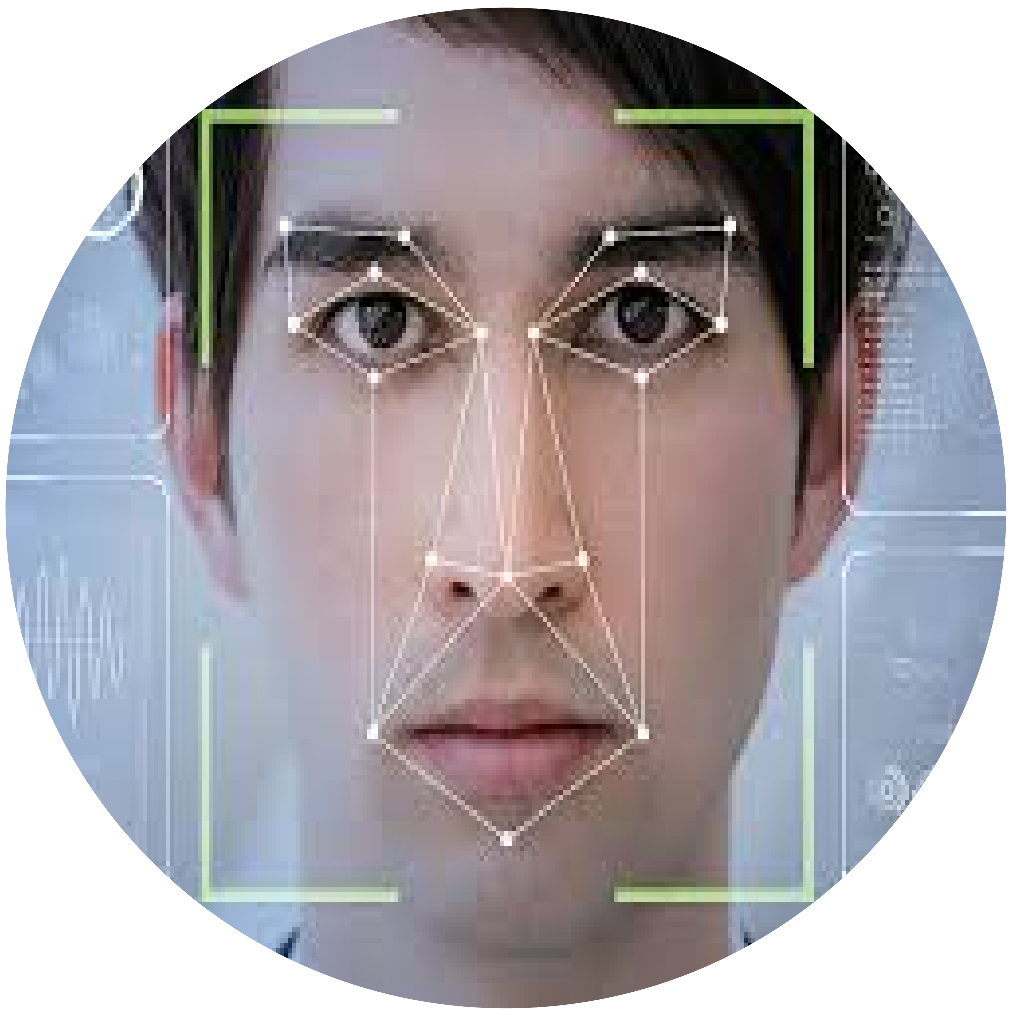

Face and Smile Detection with OpenCV

15.10.2019

Computer Vision

Face and Eye Detection

```python
# Face Recognition

import cv2

face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml') # We load the cascade for the face.
eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml') # We load the cascade for the eyes.

def detect(gray, frame): # 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.
    faces = face_cascade.detectMultiScale(gray, 1.3, 5) # We apply the detectMultiScale method from the face cascade to locate one or several faces in the image.
    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.
        eyes = eye_cascade.detectMultiScale(roi_gray, 1.1, 3) # We apply the detectMultiScale method to locate one or several eyes in the image.
        for (ex, ey, ew, eh) in eyes: # For each detected eye:
            cv2.rectangle(roi_color,(ex, ey),(ex+ew, ey+eh), (0, 255, 0), 2) # We paint a rectangle around the eyes, but inside the referential of the face.
    return frame # We return the image with the detector rectangles.

video_capture = cv2.VideoCapture(0) # We turn the webcam on.

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.
```

Same Code Explained in Detail

```
#Import the libraries. We are only importing openCV library for now.
import cv2

#Loading the cascades.(Trained models, one for face detection another for eye detection. Downloaded
from openCV)
#An object will be constructed.
#CascadeClassifier is a class inside cv2 module.
#Object takes only one parameter which is the xml file: xml file can be taught as a trained model.
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

#Create the eye classifier object
eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')

#Now let's define a function that will do the detections
#Function is going to be applied on a single image and then it will return the same image with a
rectangle that is detecting the faces and eyes.
#So the detect() function will get an image as an input.
#Images will be the different images of the video but the function will get an image in each time.
#Also it will return another image with rectangles if needed.
#Remember the theory part. Cascade works on black and white images.

def detect(gray,frame):
    '''
    gray: black and white image
    frame: original image

    '''
    #we want to get the coordinates of the rectangle(s) that shows the detected faces.
      #there is a method of cv2.CascadeClassifier class which returns the coordinates of the
rectangle(s) as a tuple.
    #tuples of 4 elements: x,y are the upper left coordinates of the rectangle, w is the width and
h is the height
    faces = face_cascade.detectMultiScale(gray, 1.3, 5)
    #the method get 3 inputs:
       #gray: black and white image
        #1.3: scale factor which tells how much the size of the image is going to be reduced or
equivalently how much the size of the filter is going to be increased. The size of the image will
be reduced by 1.3 times.
        #5: minimum number of neighbours. In order for a zone of pixels to be accepted, you need
have at least certain number of neighbour zones that are also accepted.

    """
    *scaleFactor – Parameter specifying how much the image size is reduced at each image scale.
    *Basically, the scale factor is used to create your scale pyramid.
    *More explanation, your model has a fixed size defined during training, which is visible in the
XML.
     This means that this size of the face is detected in the image if present.
      However, by rescaling the input image, you can resize a larger face to a smaller one, making
it detectable
     by the algorithm.

     *1.05 is a good possible value for this, which means you use a small step for resizing, i.e.
reduce the size by 5%,
    you increase the chance of a matching size with the model for detection is found.
    This also means that the algorithm works slower since it is more thorough.
    You may increase it to as much as 1.4 for faster detection, with the risk of missing some faces
altogether.
```

```
     *minNeighbors – Parameter specifying how many neighbours each candidate rectangle should have
to retain it.
    *This parameter will affect the quality of the detected faces.
    Higher value results in fewer detections but with higher quality. 3~6 is a good value for it.
    """

    #now we have the coordinates of the rectangles of detected faces.
    #let's iterate through the faces tuple and draw actual rectangles and detect eyes inside these
rectangles.

    for (x,y,w,h) in faces:
        #now we have the coordinates of a rectangle
        #let's draw an actual rectangle using a method inside the openCV module:
        cv2.rectangle(frame, (x,y), (x+w,y+h), (255,0,0), 2)
        #4 inputs
           #frame:  original image
           #(x,y): a couple of the coordinates of the upper-left corner
             #(x+w,y+h) coupe of the coordinates of the lower-right corner. COORDINATE SYSTEM IS A
NUMPY ARRAY!!!
           #RGB code of the colour we want to
           #Thickness of the edges of the rectangle(s)

        #Now to save from computations we can check eyes inside these rectangles.
        #First we will get 2 regions of interests inside the rectangles
        #One ROI for b&w image in which the cascade is going to be applied to detect the eyes.
          #Another ROI for the original image because we'll draw the rectangles in the original
image.

        roi_gray = gray[y:y+h, x:x+w] # corresponds to the zone inside the detector rectangle.
         roi_color = frame[y:y+h, x:x+w] # x and y reversed because probably since gray and frame
are arrays the vertical axis (y) actually should be written first.

        #now let's detect the eyes inside the roi_gray
        eyes = eye_cascade.detectMultiScale(roi_gray, 1.1, 3)

        #now let's draw the rectangles for the eyes inside another for loop
        for (ex,ey,ew,eh) in eyes:
            cv2.rectangle(roi_color, (ex,ey), (ex+ew,ey+eh), (0,255,0), 2)


        #now return the original image with rectangles detecting the face(s) and eyes
    return frame



#Now let's detect some faces using the webcam

#Let's get the last frame coming from the webcam. In order to get it we need to create an object
first.
video_capture = cv2.VideoCapture(0) # object created from the VideoCapture class of the cv2 module.
#input is 0 if it is a webcam and 1 if it is an external device.
#this videp_capture object contains the last frame coming from the webcam.



#Now since we have the last webcam frame let's apply the detect() method on it inside an infinite
loop:
```

```
while True:
    #now let's use the read method of the video_capture object.
    #it returns two elements by we only need the second one which is the last frame
    #thus we can only obtain the second return by using:
    _, frame = video_capture.read()

    #obtain the gray image from the original image:
    gray = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)


      #let's apply the detect function. Returned image with rectangles is assigned into canvas
variable.
    canvas = detect(gray,frame)


    #Display all the successive outputs (canvases) in an animated way:
    cv2.imshow('Video',canvas)

    #Let's assign a quit key as q:
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

#Now the while loop is over then we should turn off the webcam:
video_capture.release()
#Get rid of the window that canvases are animated.
cv2.destroyAllWindows()
```

Example Output for Face and Eye Detection

This GIF is taken from: http://procodetech.com/portfolio/face-eye-smile-detection/

Smile Detection Code

```
# Smile Recognition

# Importing the libraries
import cv2

# Loading the cascades
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
smile_cascade = cv2.CascadeClassifier('haarcascade_smile.xml')

# Defining a function that will do the detections
def detect(gray, frame):
    faces = face_cascade.detectMultiScale(gray, 1.3, 10)
    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]
        smiles = smile_cascade.detectMultiScale(roi_gray, 1.3, 22)
        for (sx, sy, sw, sh) in smiles:
            cv2.rectangle(roi_color, (sx, sy), (sx+sw, sy+sh), (0, 255, 0), 2)
     return frame #Bu çalışıyor çünkü roi_color üzerinde yapılan her değişiklik frame üzerinde de
yapılıyor.

# 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()
```

More blogs