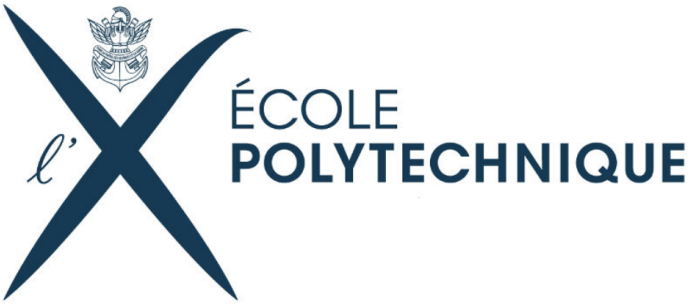

Ecole Polytechnique

INF-573: Image Analysis and Computer Vision

Face detection and anonymisation in real time video

Professor:
Mathieu Brédif
Vicky Kalogeiton

Group:
Naomie Halioua
Jules Arbelot

December 15, 2020

Contents

1 Introduction

1.1 Motivations

We live in an era where privacy is at the center of our concerns especially when it comes to the numerical revolution. People are more and more concerned with their web image and the privacy of their live. The General Data Protection Regulation of 2018 really shows that giving control to individuals over their personal data was a very serious topic at least in the EU and is something technology will have to adapt to.

In addition to that, because of the COVID epidemic, people are more and more using numerical tools to communicate for work. Zoom, Skype, Microsoft Teams are now common place in most companies. They are used for teleconferencing, telecommuting, distance education or even simple social relations. These soft wares have faced public scrutiny related to security and privacy issues and even nowadays, some people or companies are still mistrustful.

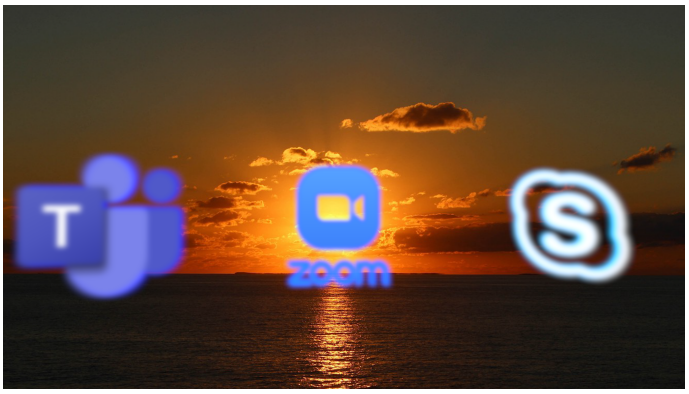

Figure 1: Major companies in teleconferencing and social relations

This is where anonymization of videos comes in. People want and need more tools to efficiently protect their privacy during teleconferencing or social relations. We decided to work on a project where we try to make an easy to use tool to make a video anonymous during live teleconferencing. It is aiming to help people working with sensitive information, people that are making anonymous interviews or testimonials or people talking of personal business. We thought it was interesting to see what we could do to make a real-time video anonymous.

1.2 Anonymous Video Softwares

As we said earlier, anonymizing a video is a booming problem. We found that lots of companies have developed their own anonymisation software. For example Anonymous Camera is an app developed by the London AI startup Playground that provides an easy way for interviewers to make anonymous video. The founders of the startup explained that they where inspired to create Anonymous Camera after reading reports of journalists in the United Arab Emirates who were writing about the persecution of LGBT groups.

"It's very important that whenever you take a photo of someone you have to be very careful to anonymize that immediately", "People capture footage and then anonymize it through Photoshop, but in the meantime the government might have confiscated their camera, and with that the compromising material." says the creators of the app. Anonymous Camera removes the need for this entire post-production process because it creates directly an anonymous video. There is no original one.

Also, we knew we were not aiming to do this software for journalism but for our fellow students of Ecole Polytechnique and other schools. The idea behind the project was to make anonymization fun and simple for everyday users of Zoom for academic purpose. In the end, this project answers a major issue in today's world but is also a way to make a funny algorithm that can be used by students and friends.

1.3 How to make video anonymization original ?

We first started to look at what journalists or softwares usually do to anonymize a video. The first thing to do is to hide the face to make it not recognizable. Lets see some examples:

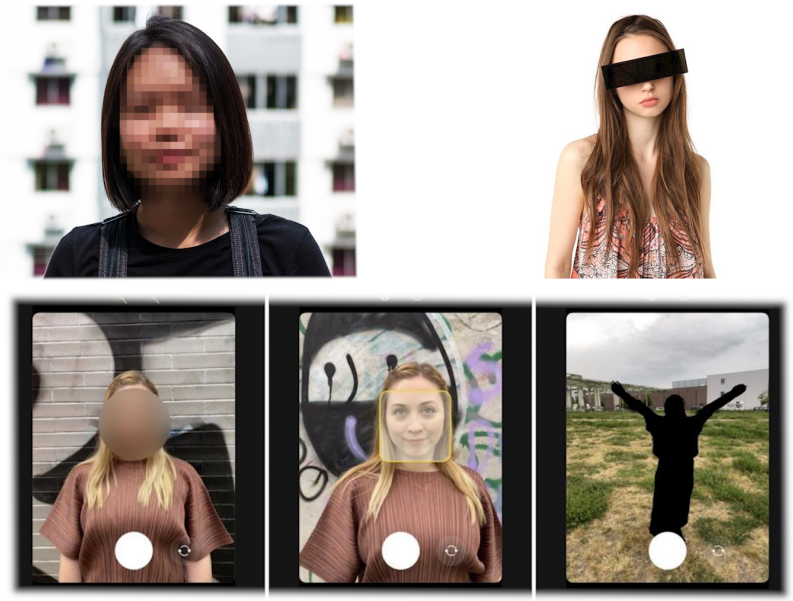

Figure 2: Examples of video anonymisation

The first image shows the first idea that we had which was pixelating or blurring the face. The second image shows another option that we see very often in newspapers. The image we choose to show comes from an article in "Le Monde" which explains how it was far less efficient than covering the hole face. Finally, we decided to show you the three options of Anonymous Camera. the first

one is to draw a blurred circle on top of the face, the second is to put a screen in front of the face with controllable properties and the third one displays a black color matte on the body and face of the person being filmed.

From that, realised that if we wanted our software to be nice and cool to use we needed to add different options to it, just like Anonymous Camera but maybe some funny ones. Obviously the first step is to blur the face because it is the simplest and most efficient method. But after that, we could try different methods. It was also important for us to make something funny to use, that would not make anonymisation something weird or serious. Look at for example how Zoom allows you to hide your room with a background. It is nice for everyone and doesn't put to much attention on the fact that maybe you are doing it because you don't want your private place being seen by others. Also, the image of the activist group Anonymous inspired us to make "masks" to hide the face in a funny way.

2 Our Project

2.1 Face detection and eyes detection using OpenCV

In this section, we are going to use OpenCV to do real-time face detection and anonymisation from a live stream via our webcam. We have decided to also detect the eyes for a more advanced anonymisation process.

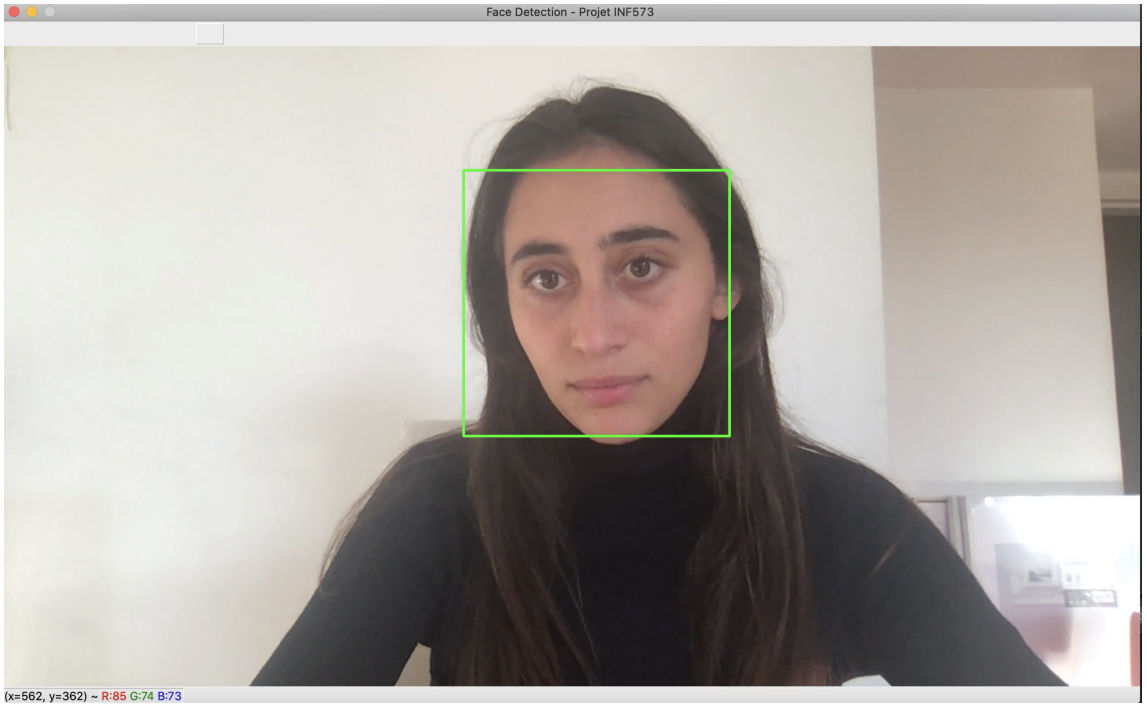

Figure 3: Examples of Face Detection

Videos are made up of frames, which are still images. We perform the face detection for each frame in a video. So when it comes to detecting a face in still image and detecting a face in a real-time video stream, there is not much difference between them.

2.1.1 Haar Cascade algoritm

The first step to anonymize our videos was to track the face of the subject. To do so, we used the **Haar Cascade object** detector. We used the pretrained models already present in OpenCV to detect faces and eyes. Instead of creating and training the model from scratch, we use trained Haar Cascade models which are saved as XML files.
First we load and configure HaarCascade Classifiers. After loading the classifier, we open the webcam using a simple OpenCV one-line code

```
import os
# location of OpenCV Haar Cascade Classifiers:
cascPath = os.path.dirname( cv2.__file__) + "/data/haarcascade_frontalface_alt2.xml"
cascPath_eye = os.path.dirname( cv2.__file__) + "/data/haarcascade_eye.xml"
# build our cv2 Cascade Classifiers
faceCascade = cv2.CascadeClassifier(cascPath)
eye_cascade = cv2.CascadeClassifier(cascPath_eye)
# collect video input from first webcam on system
cap = cv2.VideoCapture(0)
```

We first followed the basic OpenCV tutorial of Cascade Classifiers. We used the *detectMultiScale* function to detect the faces. The parameters passed to *detectMultiScale* help determine the minimum size of faces that can be detected in the image, how close faces can be together before they can't be detected, and other options. We choose to tweak the parameters to these numbers because they feel appropriated for webcam videos and real-time computing.
We also create greyscale image from the video feed. The black and white image is used to face detection, the color version is the one we will return.

```
while True:
    # Capture video feed
    ret,frame = cap.read()

    if(ret):

        # Create greyscale image from the video feed
        gray = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
        frame = cv2.cvtColor(frame,cv2.COLOR_BGR2BGRA)

        # Detect faces in input video stream
        f = faceCascade.detectMultiScale(gray, scaleFactor=1.1,minNeighbors=7)
```

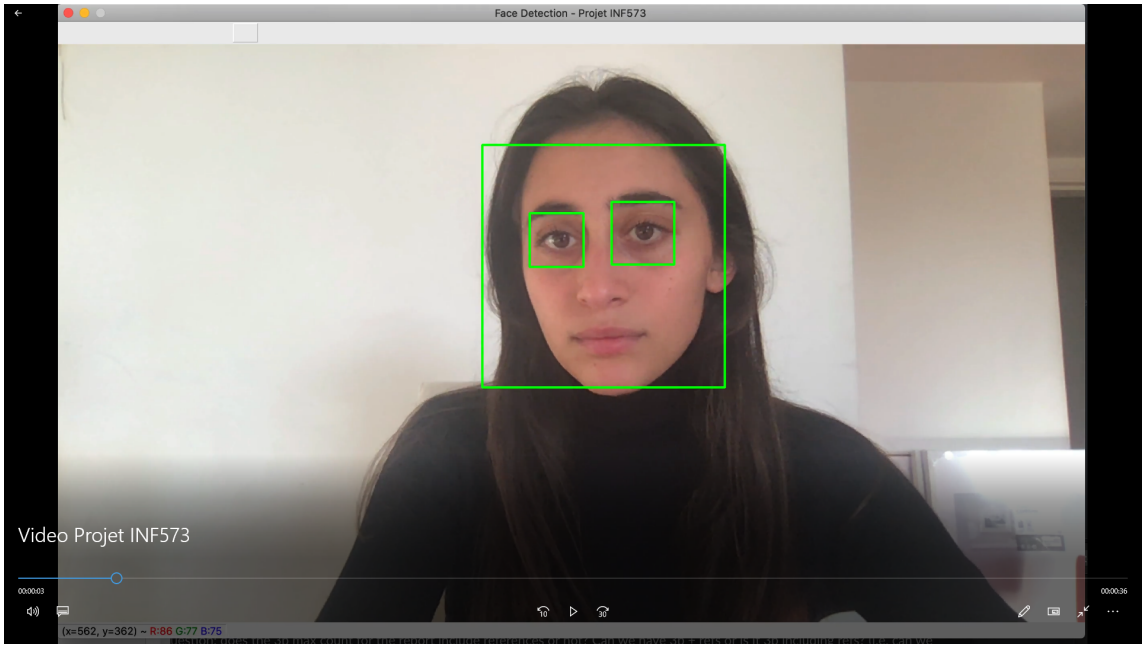

Figure 4: Examples of Eyes Detection

2.1.2 Face and Eyes Detection

After setting up the HaarCascade Detector our next step is to break down the faces first, before getting to the eyes(because we can't have faces without eyes or eyes without faces) .Here, we're finding faces, their sizes, drawing rectangles, and noting the Region Of Interest (ROI), next, we poke around for eyes.

```
# Iterate over each face found
        for x,y,w,h in f:
            #region of interest of each frame
            roi_gray = gray[y:y+h, x:x+w]
            roi_color = frame[y:y+h, x:x+w]

           # Detect eyes in input video stream
            eyes = eye_cascade.detectMultiScale(roi_gray, scaleFactor=1.35,
            minNeighbors=6)
```

The variable faces now contain all the detections for the target image. Detections are saved as pixel coordinates. Each detection is defined by its top-left corner coordinates and width and height of the rectangle that encompasses the detected face.

To show the detected face, we will draw a rectangle over it.OpenCV's rectangle() draws rectangles over images, and it needs to know the pixel coordinates of the top-left and bottom-right corner. The

coordinates indicate the row and column of pixels in the image. We can easily get these coordinates from the variable face. Rectangular frame is created using *cv2.rectangle* of green colour.

```
if framed:
                # Draw rectangle around the faces
                cv2.rectangle(frame,(x,y),(x+h,y+w),(0,255,0),2)

if framed_eyes:
                #Iterate over each eyes found
                for (ex,ey,ew,eh) in eyes:
                    cv2.rectangle(roi_color,(ex,ey),(ex+ew,ey+eh),(0,255,0),2)
```

From that we had a 4 point detection of the face and the eyes that we could use to blur the square of the face.

2.2 Bluring techniques to anonymize faces

Image blurring is achieved by convolving the image with a low-pass filter kernel. OpenCV provides four main types of blurring techniques.**Gaussian Blurring, Median Blurring,Averaging** and **Bilateral Filtering**. We implement in our projet the first two one.

2.2.1 Gaussian Blur

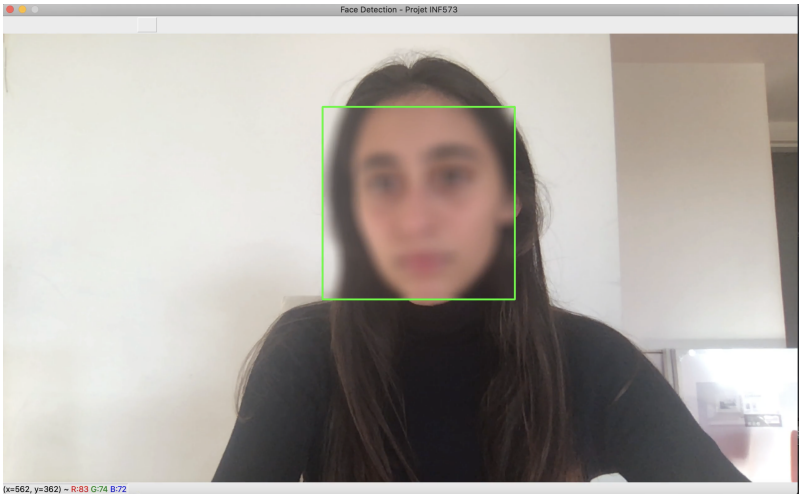

Figure 5: Gaussian blurred face test

We first used Gaussian Blur on the square because it's the simple blur to compute and the blurring method we had seen the most during the course. It is done with the function, *cv.GaussianBlur()*. We should specify the width and height of the kernel which should be positive and odd. We also should specify the standard deviation in the X and Y directions, sigmaX and sigmaY respectively. Gaussian blurring is highly effective in removing Gaussian noise from an image.

Once we find the ROI, we can blur it using *cv2.GaussianBlur*. We just have to tell which region of the image has to be blurred: the part that contains the faces. Then, we assign the blurred portion of the image to the complete frame.

```
if blur_faced:
    # apply a gaussian blur on this new rectangle image
    roi_color = cv2.GaussianBlur(roi_color,(23, 23), 30)
    # merge this blurry rectangle to our final image
    frame[y:y+roi_color.shape[0], x:x+roi_color.shape[1]] = roi_color
```

2.2.2 Median Blur

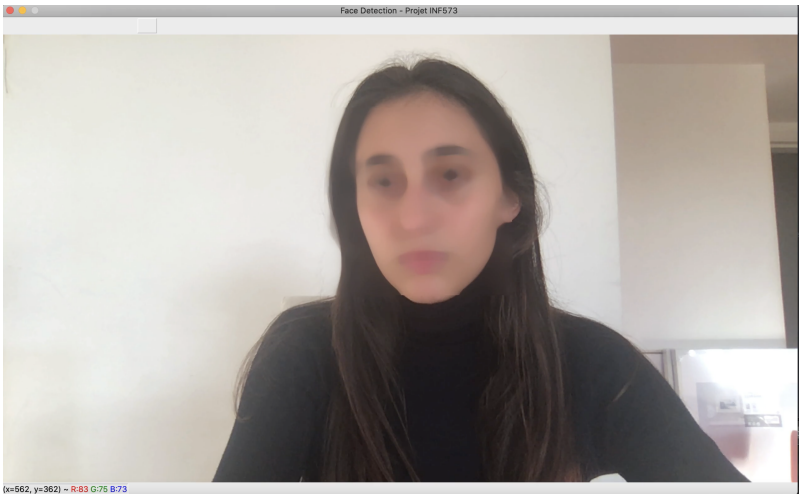

Figure 6: Median blurred face test

Since this blur was not very pretty and natural, we wanted to add a blur that would add personality to the video. So we implemented a Median Blur filter. This created much more "smooth" blurs that are keeping the globalform and lines of the face but just make it not recognizable

```
if blur_faced_2:
                # apply a median blur on this new rectangle image
                roi_color = cv2.medianBlur(roi_color,23)
                # merge this blurry rectangle to our final image
                frame[y:y+roi_color.shape[0], x:x+roi_color.shape[1]] = roi_color
```

Now it has this smooth impression of still having a human person but with a face you can't recognize in theory. The function *cv2.medianBlur()* takes the median of all the pixels under the kernel area and the central element is replaced with this median value.

3 Adding options

Now that we had a clean blurred face we wanted to add many real-time options to make the anonymization more fun.

3.1 Eyes

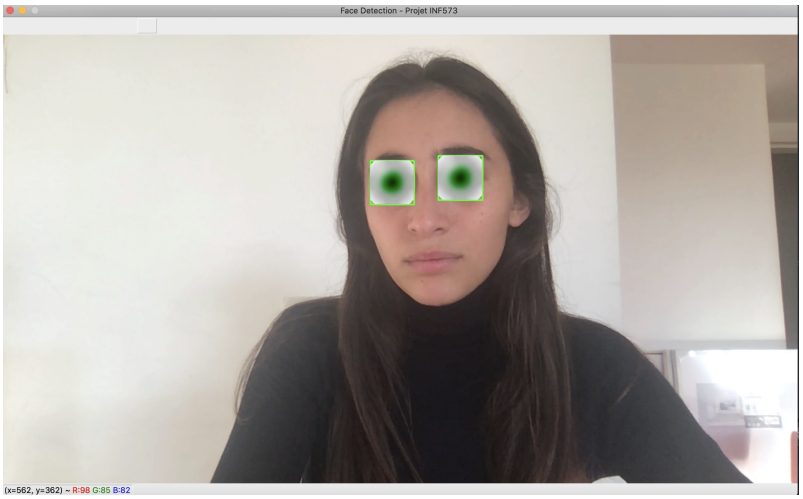

Figure 7: Moustached face

Since we have implemented eye detection, we will use this to add an eye filter. It's true that it's far from being real, but let's not forget that we are here to anonymize and not change a person's identity.

```
if glasse:
    #Iterate over each eyes found
    for (ex,ey,ew,eh) in eyes:
        cv2.rectangle(roi_color,(ex,ey),(ex+ew,ey+eh),(0,255,0),2)
        roi_eyes = roi_gray[ey:ey + eh, ex:ex + ew]
        # Resize the original image
        gls = cv2.resize(gls.copy(),(ew, int(ew/filter_size_ratio_glasse)))

        for i in range(gls.shape[0]):
        for j in range(gls.shape[1]):
            if gls[i,j][3] != 0:
                roi_color[ey+i, ex+j] = gls[i,j]
```

In OpenCV we will often talk about Regions of Interest, or ROI. When modifying an image, we often will only modify a small section of the image. Rather than making changes to the entire image, we will select a region from the original image, where the coordinate system for the region is relative to the region, rather than the image. Here the ROI in eyes detection is obviously different from the ROI in face detection.
Following our idea to make a creative anonymization program, we added a few other features like squares over the eyes, etc. It's works with the same basic principles as the eyes. It's also possible to combine two filter. We will see how more in details below.

Figure 8: Moustached face

3.2 Moustache

As we said earlier, we wanted to make this moustache because of how fun it was and because it is a way to anonymize a video without a creepy blur or black stripes.

```
if mustache:
                    # w as face width h as face height
                    mst_width = int(w*0.7)+1
                    mst_height = int(h*0.2)+1

                    # Resize the original image
                    mst = cv2.resize(mst,(mst_width,mst_height))

                    for i in range(int(0.7*h),int(0.7*h)+mst_height):
                        for j in range(int(0.2*w),int(0.2*w)+mst_width):
                            for k in range(3):
                                if mst[i-int(0.7*h)][j-int(0.2*w)][k] < 235:
                                    frame[y+i][x+j][k] = mst[i-int(0.7*h)][j-int(0.2*w)][k]
```

First we load the mustache with -1 as the second parameter to load all the layers in the image. The image is made up of the 4 traditional layers. We take just the alpha layer and create a new single-layer image that we will use for masking. The mask will define the area for the mustache, and the inverse mask will be for the region around the mustache).
We keep the original mustache image sizes, which we will use later when re-sizing the mustache image. The height of the mustache with relation to the face should be proportional to the width of the mustache and the original image (maintain proportional to the original image).

Figure 9: Moustached face

How to compile ? We have designed the algorithm so that each time it iterate, it waits for you to click on a particular key to apply its filter. The "a" key corresponds to face detection while the "b" key corresponds to eye detection. We initialized variable at the beginning.

```
#Variable initialization
blur_faced = False
framed = False
mustache = False
framed_eyes = False
blur_faced_2 = False
blur_faced_3 = False
glasse = False
black = False
{main code}
if ch == ord("a"):
        framed = not framed
if ch == ord("b")
    framed_eyes = not framed_eyes
    ....
```

We also add a special order, in order to save automatically a picture of the screen in our documents:

```
if ch==ord('s'):
        count=0
        count=count+1
        name="frame"+str(count)+".jpg"
        cv2.imwrite(name,frame)
```

Then, we can superpose several filters as we can see in exemple above.

4 To go further

Here is a small paragraph explaining what we could have done if we had more time or if we wanted to make a better software.

First of all, we would have liked to run a deblur and face recognition algorithms on our blurred faces to see how efficient it is. Same thing for the moustache. We are guessing that adding the moustache and the eyes make it really hard to recognize the person with an algorithm or AI.

Also, since we are making a tool for teleconferencing users, we would have to deal with the voice too just like Anonymous Camera does it. However none of us new about sound treatment so we decided to leave that aside.

5 Conclusion

In the end, our project is a first attempt to answer today's world problems and bring the tools people need for their safety or liberty. We decided to make something adapted to our student environment so by making a fun and original anonymization software. It uses basic Computer Vision features and is a good introduction to the Image Analysis sector.

References

https://www.mygreatlearning.com/blog/real-time-face-detection/sh1
https://medium.com/@bipinadvani/face-recognition-and-blurring-in-webcam-using-cv2-python-5c4c589e6e59
https://pythonprogramming.net/haar-cascade-face-eye-detection-python-opencv-tutorial/
https://github.com/kurttepelikerim/FaceDetectionAnonymization/blob/master/FaceDetection.py
https://docs.opencv.org/master/d4/d13/tutorial$_{p}y_{f}$*iltering.html*
$https://github.com/kunalgupta777/OpenCV-Face-Filters$
$https://sublimerobots.com/2015/02/dancing-mustaches/$
$https://github.com/charlielito/snapchat-filters-opencv/blob/master/README.md$