ASSIGNMENT-9

NAME : Kavish Mehta
COLLEGE ID: 18BIS0130

1) Build the Simple use case to draw the shapes in the input video

For this purpose I have made a simple application where we can make a rectangle around a video. That particular region of the live video becomes sketched video

```
In [1]: #NAME: Kavish Mehta

        import cv2
        from matplotlib import pyplot as plt
        import numpy as np
```

I have imported 3 libraries, cv2 library is the opencv library used for image/video manipulation, matplotlib library is used to draw the image temporarily on it, numpy library is used to support the data of the images and videos.

```
In [2]: def sketch_transform(image):
            image_grayscale = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
            image_grayscale_blurred = cv2.GaussianBlur(image_grayscale, (7,7), 0)
            image_canny = cv2.Canny(image_grayscale_blurred, 10, 80)
            _, mask = image_canny_inverted = cv2.threshold(image_canny, 30, 255, cv2.THRESH_BINARY_INV)
            return mask
```

The above function is used to convert region of interest into sketch based video

```
In [3]: cam_capture = cv2.VideoCapture(0)
        cv2.destroyAllWindows()

        while True:
            _, im0 = cam_capture.read()
            showCrosshair = False
            fromCenter = False
            r = cv2.selectROI("Image", im0, fromCenter, showCrosshair)
            break
```

In the above lines I capture one frame from the live video. This frame is displayed on a screen derived from matplotlib library.

selectROI() function helps us to draw a rectangle and select the Region Of Interest.

```
while True:
    _, image_frame = cam_capture.read()

    rect_img = image_frame[int(r[1]):int(r[1]+r[3]), int(r[0]):int(r[0]+r[2])]

    sketcher_rect = rect_img
    sketcher_rect = sketch_transform(sketcher_rect)

    #Conversion for 3 channels to put back on original image (streaming)
    sketcher_rect_rgb = cv2.cvtColor(sketcher_rect, cv2.COLOR_GRAY2RGB)

    #Replacing the sketched image on Region of Interest
    image_frame[int(r[1]):int(r[1]+r[3]), int(r[0]):int(r[0]+r[2])] = sketcher_rect_rgb

    cv2.imshow("Sketcher ROI", image_frame)

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

cam_capture.release()
cv2.destroyAllWindows()
```

In the above code i capture the live video constantly, rect_img stores the coordinates for ROI, sketcher_rect variable stores the sketched version of the particular part of video.

OUTPUT :

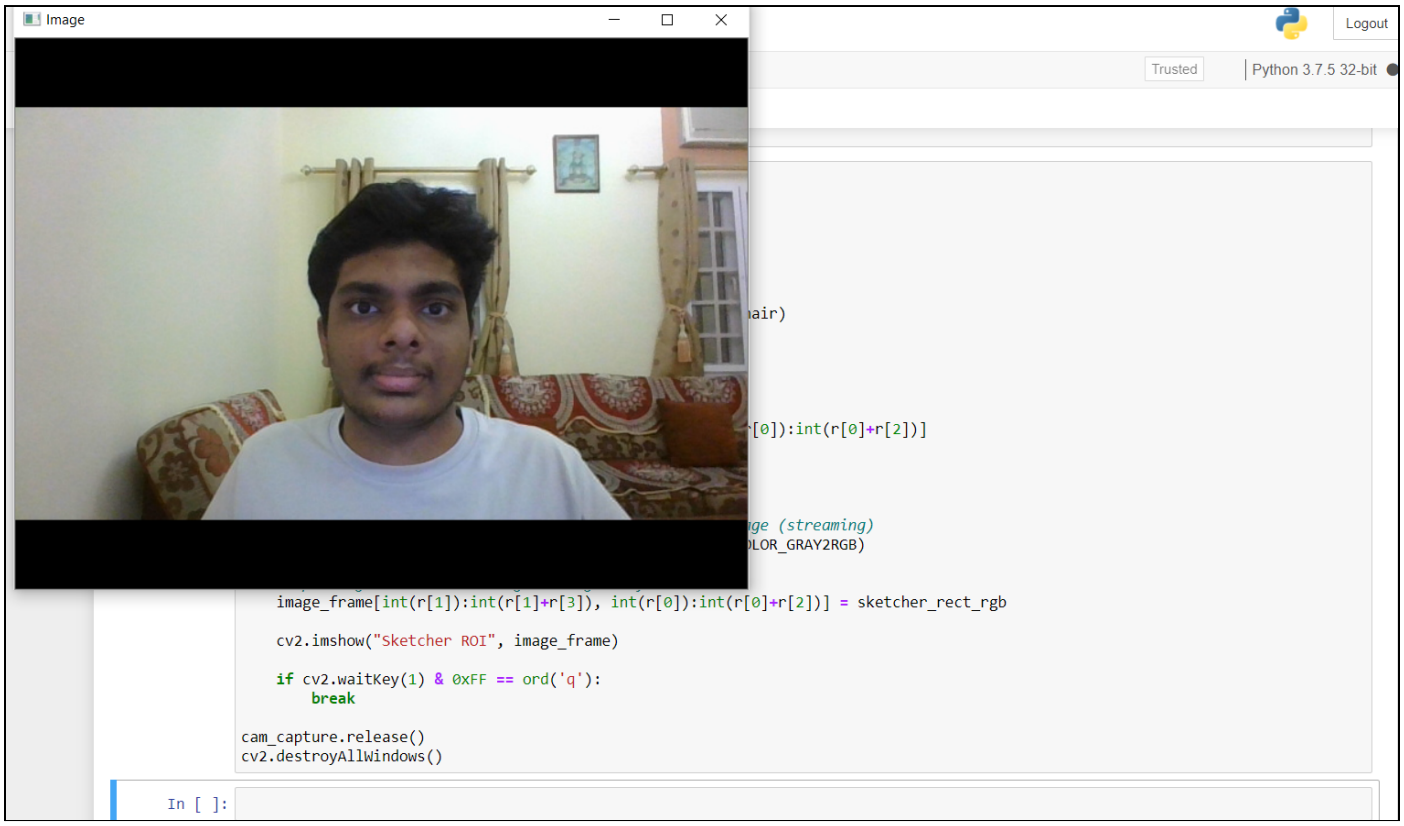
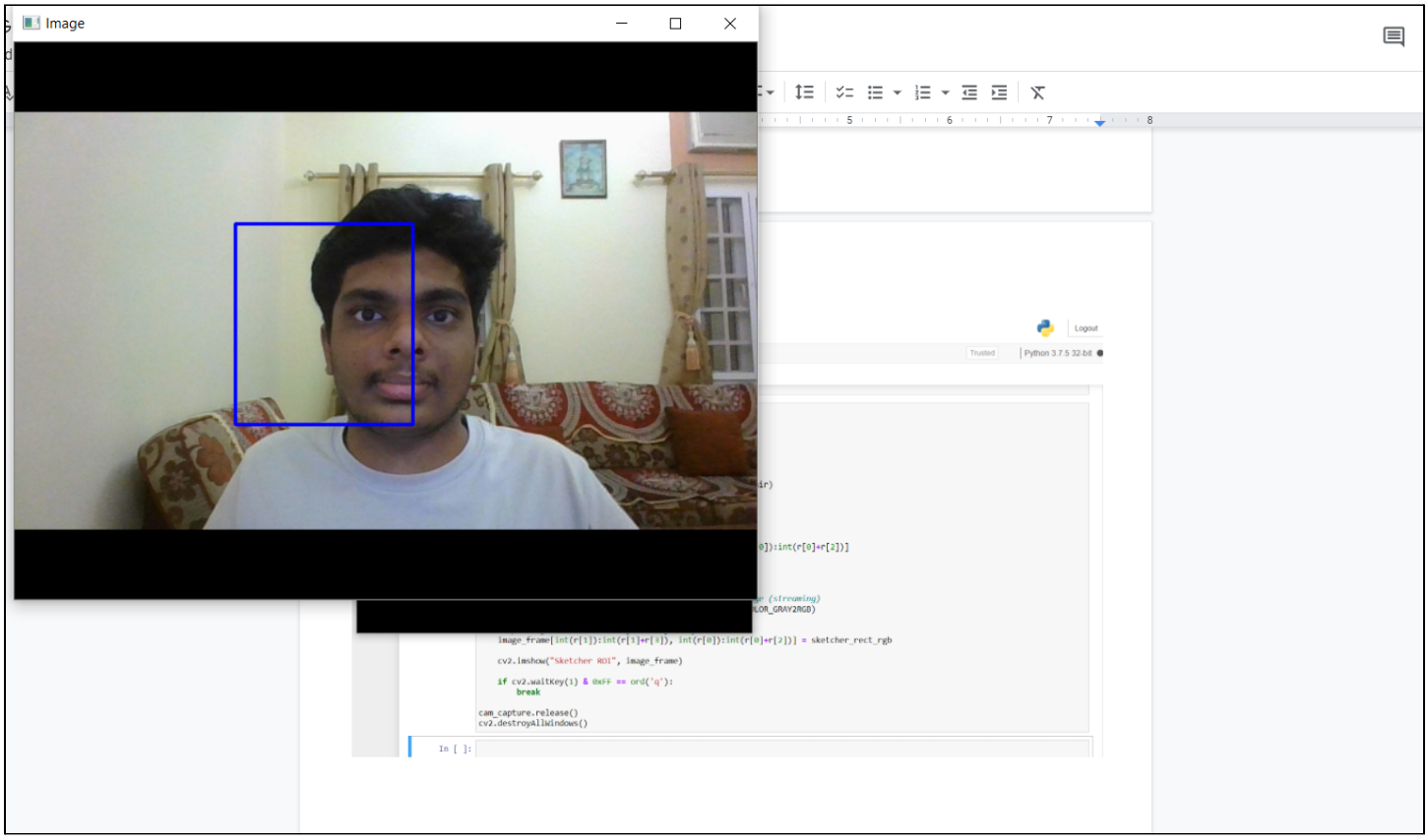


Image
Logout
Trusted
Python 3.7.5 32-bit
image_frame[int(r[1]):int(r[1]+r[3]), int(r[0]):int(r[0]+r[2])] = sketcher_rect_rgb
cv2.imshow("Sketcher ROI", image_frame)
if cv2.waitKey(1) & 0xFF == ord('q'):
break
cam_capture.release()
cv2.destroyAllWindows()
In [ ]:



Image

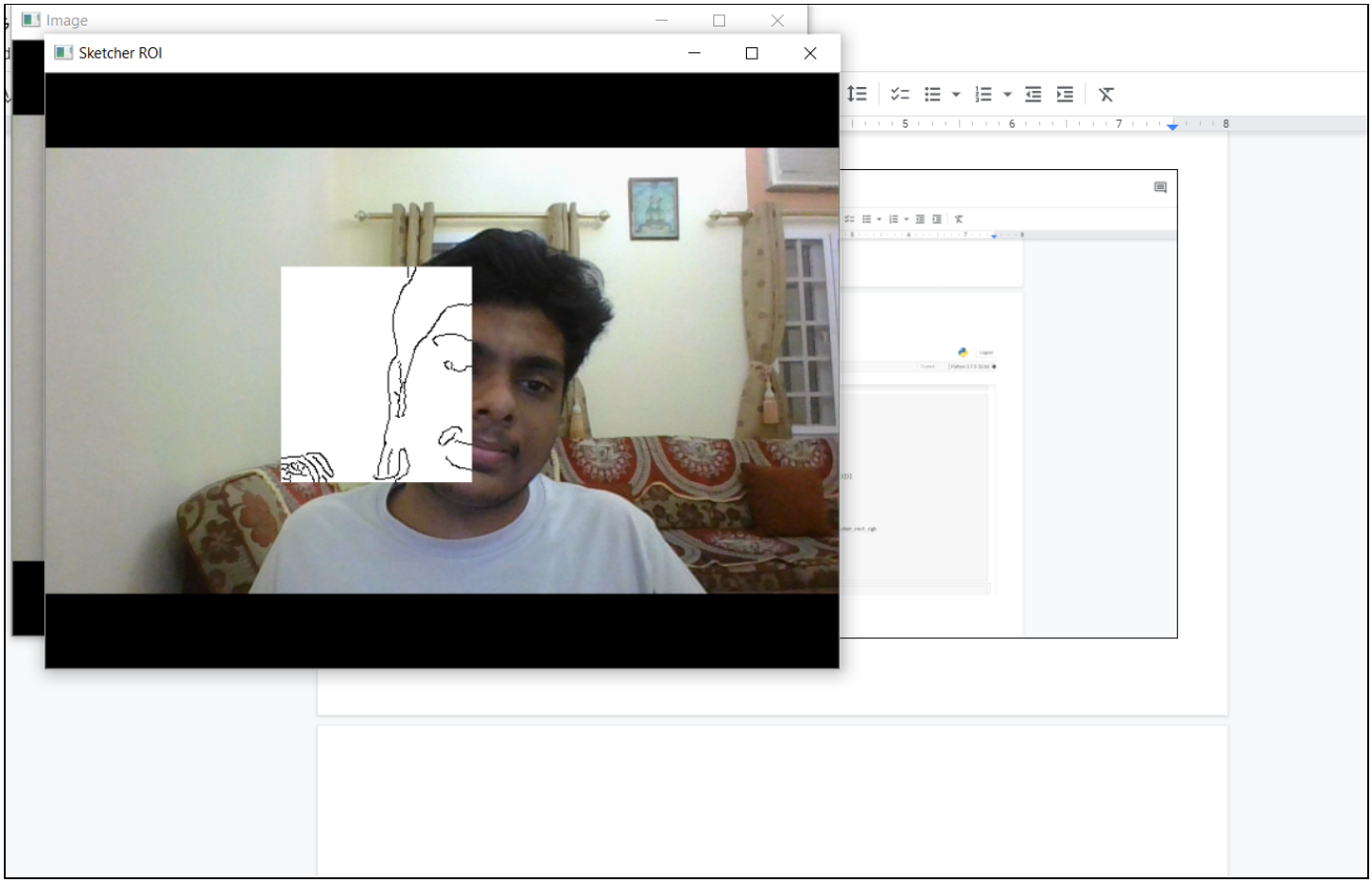
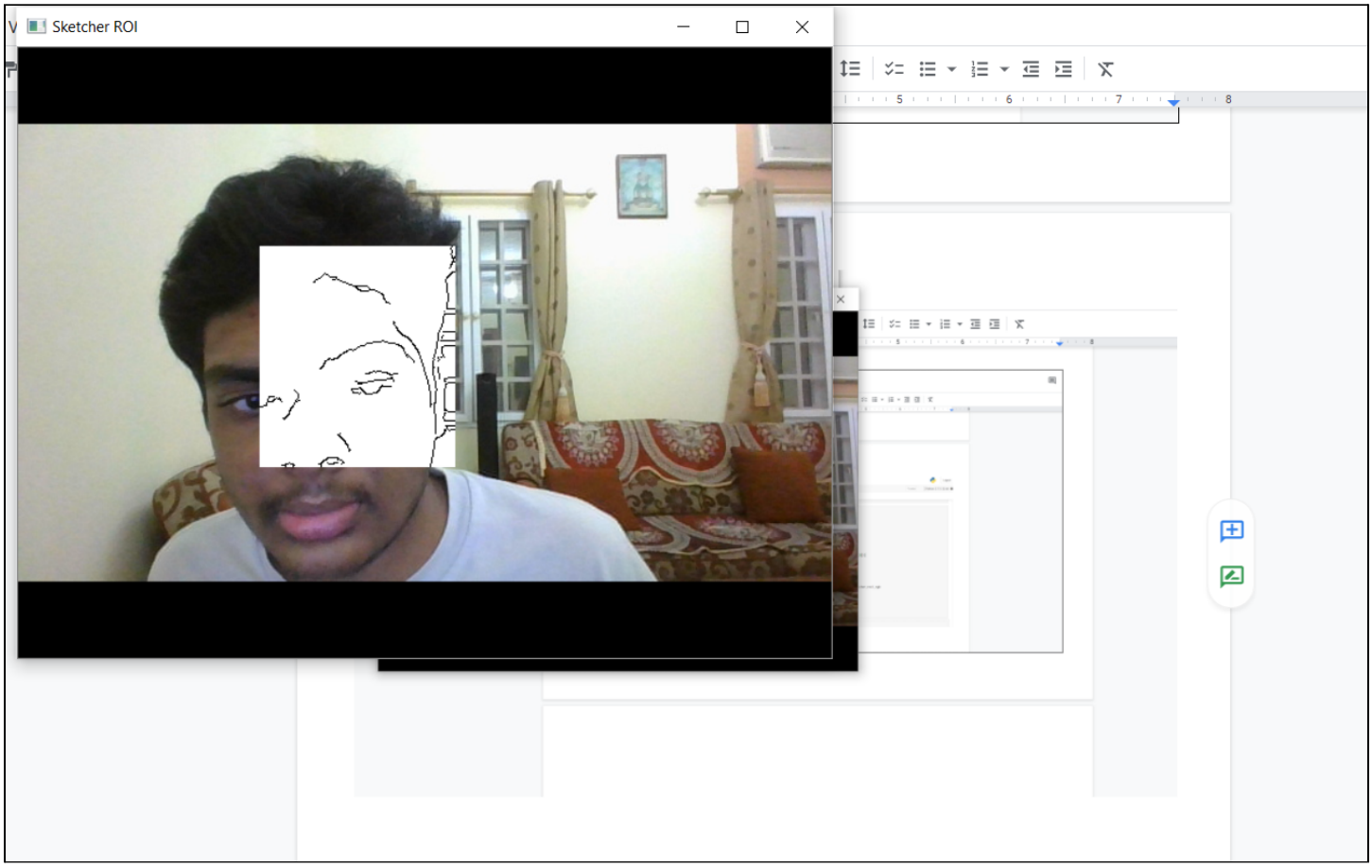


Image
Sketcher ROI


Sketcher ROI

2) Build the face detection application using OpenCV

```
In [10]: #NAME: Kavish Mehta

         import cv2
         import numpy as np
```

Importing libraries

```
In [11]: face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

         cap = cv2.VideoCapture(0)
```

In the first line face_cascade stores the pre-trained data of faces trained using Haarcascades object detection algorithm and stored in the form of xml file.

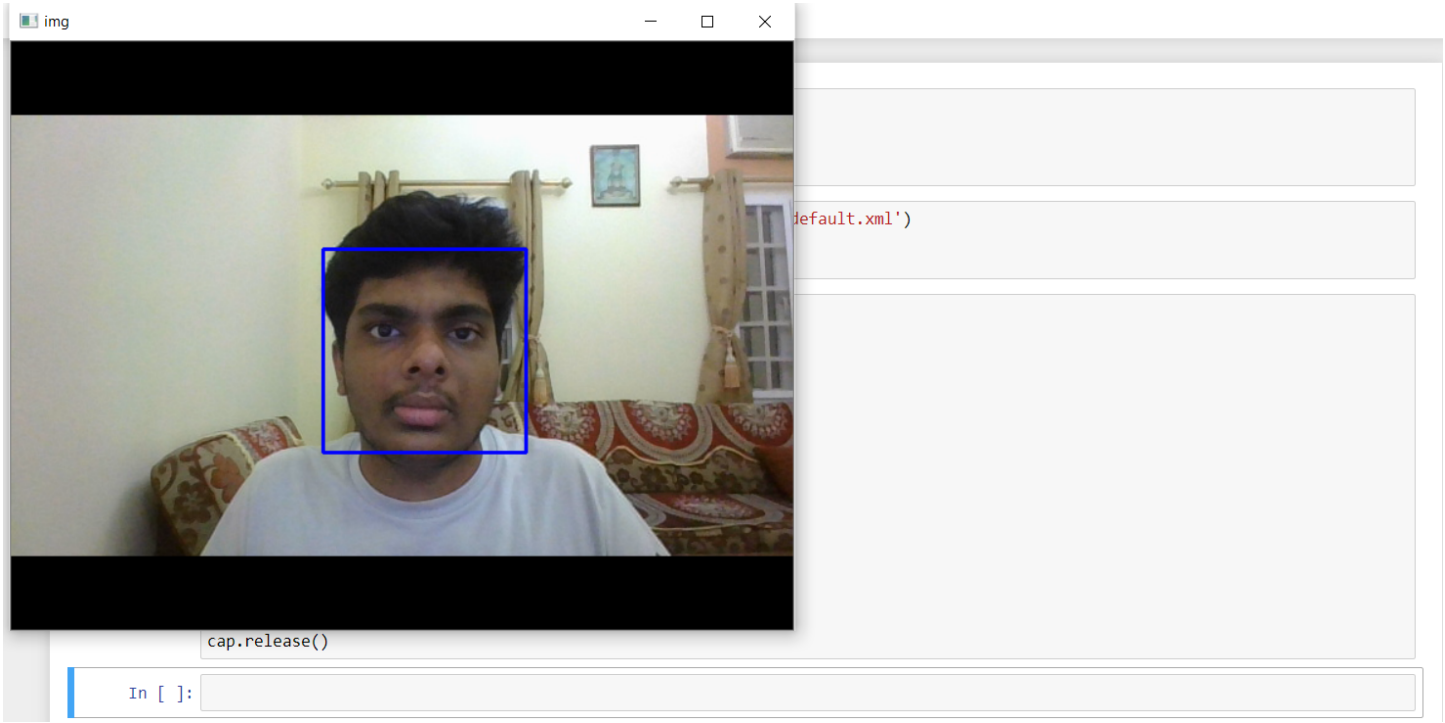
Cap stored the video capture

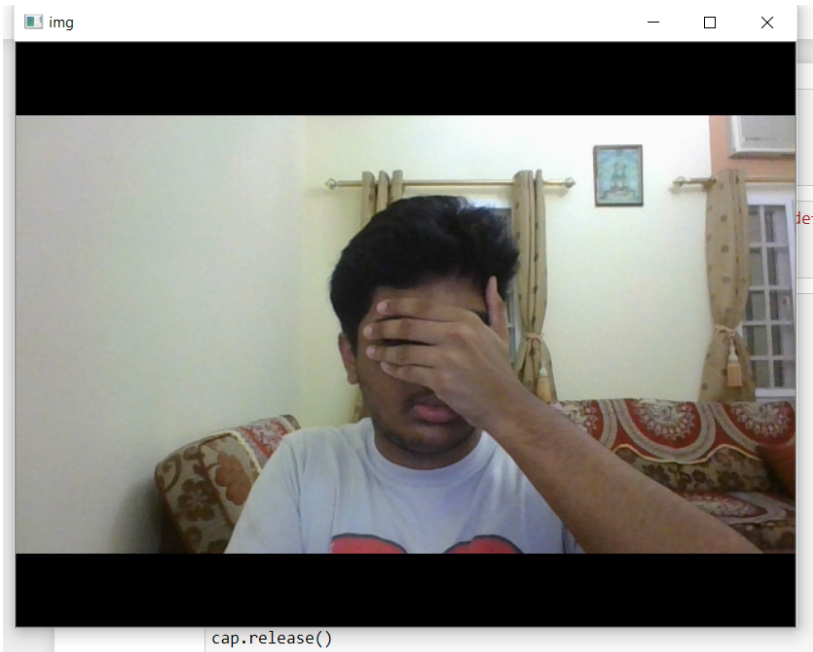
```
In [12]: while True:
             # Read the frame
             _, img = cap.read()
             # Convert to grayscale
             gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
             # Detect the faces
             faces = face_cascade.detectMultiScale(gray, 1.1, 4)
             # Draw the rectangle around each face
             for (x, y, w, h) in faces:
                 cv2.rectangle(img, (x, y), (x+w, y+h), (255, 0, 0), 2)
             # Display
             cv2.imshow('img', img)
             # Stop if 'q' key is pressed
             if cv2.waitKey(1) & 0xFF == ord('q'):
                 break
         # Release the VideoCapture object
         cap.release()
```

In the above image, the gray variable stores the grayscale version of the received frames, this is necessary since the algorithm was trained using grayscale images.

detectMultiScale function is used to detect the faces. It takes 3 arguments – the input image, scaleFactor and minNeighbours. scaleFactor specifies how much the image size is reduced with each scale. minNeighbours specifies how many neighbors each candidate rectangle should have to retain it.
faces contains a list of coordinates for the rectangular regions where faces were found. We use these coordinates to draw the rectangles in our image.

RESULTS:






```
default.xml')
```

```
cap.release()
```

In [ ]: