Face Detection using OpenCV

Objective:

- Real time detection of faces and eyes in a video file or from a Webcam by implementing the gained basic knowledge of OpenCV library.

Tools used:

- OpenCV
- Visual Studio
- C++ Programming language

Initial Setup:

- OpenCV must be properly set up in the Visual Studio for the project.
- Paths to pre-trained XML classifiers must be given to the program before execution.
- Also, to store the detected faces, path must be provided before program execution.
- Proper Webcam setup for detecting face through live stream.

Description:

This is a self-project carried out to implement the OpenCV basics. This program uses the OpenCV library to detect faces and eyes in a live stream from webcam or in a video file stored in the local machine. This program detects faces in real time and tracks it. It uses pre-trained XML classifiers for the same. The classifiers used in this program have facial features trained in them. They use the Harrcascades multilevel method detecting faces or other objects.

The detected faces are highlighted by a red coloured bounding circle and the eyes are highlighted by a blue colour bounding circle and the detected different faces can be stored at the specified location inside the system (which must be given as input before execution of program).

Explanation of Code:

Inside the main, each image from the video will be read and passed onto the detect image function. Inside this function, the image is converted from BGR to Grayscale and the faces are detected from the grayscale image and the bounding coordinates of each face is stored into a vector of rectangles.

A threshold area is set to remove errors / filter the detected objects. This threshold area can be manually changed depending upon the size of face to be detected.

According to aspect ratio, bounding circle of red colour is drawn around the face and a bounding circle of blue colour is drawn around each of the eyes on the source image which will be displayed in real time.

Also, a copy of each face image is stored in the path specified.

Implementation:

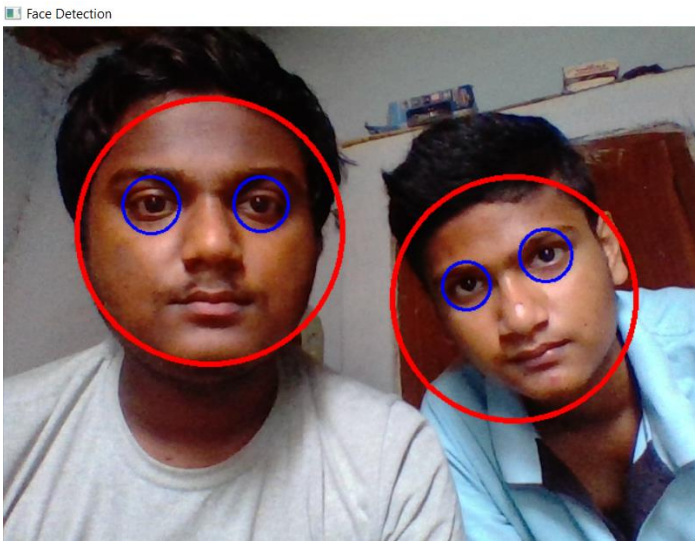
```
#include<opencv2/objdetect.hpp>
#include<opencv2/highgui.hpp>
#include<opencv2/imgproc.hpp>
#include <opencv2/objdetect.hpp> //header file to work with haarrcascade file

#include<iostream>

using namespace std;
using namespace cv;

/*PROJECT --> Detect face and eyes from a video / WebCam and stores different faces*/

void detectImage(Mat& imgsrc, CascadeClassifier& faceCascade, CascadeClassifier& eyesCascade, double
scale, string& finpath)
{
	Mat imgGray;
	vector<Rect> faces; /*Stores the details of each face*/

	cvtColor(imgsrc, imgGray, COLOR_BGR2GRAY);
	resize(imgGray, imgGray, Size(), (1 / scale), (1 / scale), INTER_LINEAR); /*Resize the
	Grayscale Image*/
	equalizeHist(imgGray, imgGray);

	/*Detect faces of different sizes using cascade classifier*/
	faceCascade.detectMultiScale(imgGray, faces, 1.1, 2);

	for (int i = 0; i < faces.size(); i++)
	{
		Rect rect = faces[i];
		double area = faces[i].area();
		if (area < 12000)	continue; /*To remove filters*/
		cout << "Area of face " << (i + 1) << " is: " << area << endl;
		Mat currimg = imgsrc(rect);
		Point center;
		Scalar faceColor = Scalar(0, 0, 255); /*Color to draw around face --> Red */
		Scalar eyeColor = Scalar(255, 0, 0); /*Color to draw around eyes --> Blue */
		double rad;

		double aspect_ratio = (double)rect.width / rect.height; /*Finding aspect ratio of the
		detected client*/
		if (0.75 < aspect_ratio && aspect_ratio < 1.3)
		{
			/*Finding the center and radius of the circle to be drawn around the face*/
			center.x = cvRound((rect.x + rect.width * 0.5) * scale);
			center.y = cvRound((rect.y + rect.height * 0.5) * scale);
			rad = cvRound((rect.width + rect.height) * 0.25 * scale);
			circle(imgsrc, center, rad, faceColor, 4, 8); /*Drawing cirlce around the face
			on imgsrc*/
			//circle(currimg, center, rad, faceColor, 4, 8); /*Drawing cirlce around the
			face on currimg*/
		}
		else
		{
			rectangle(imgsrc, rect, faceColor, 4, 8); /*Draws rectangle if the aspect ratio
			is not satisfied*/
		}

		if (eyesCascade.empty())	continue;
		Mat imgroi = imgGray(rect); /*roi is the current face of source image*/
		vector <Rect> eyes;
		eyesCascade.detectMultiScale(imgroi, eyes, 1.1, 2); /*Detecting eyes of the image*/

		for (size_t j = 0; j < eyes.size(); j++) /*Drawing circles around the eyes*/
		{
			Rect nr = eyes[j];
			center.x = cvRound((rect.x + nr.x + nr.width * 0.5) * scale);
			center.y = cvRound((rect.y + nr.y + nr.height * 0.5) * scale);
			rad = cvRound((nr.width + nr.height) * 0.25 * scale);
			circle(imgsrc, center, rad, eyeColor, 2, 8, 0);
```

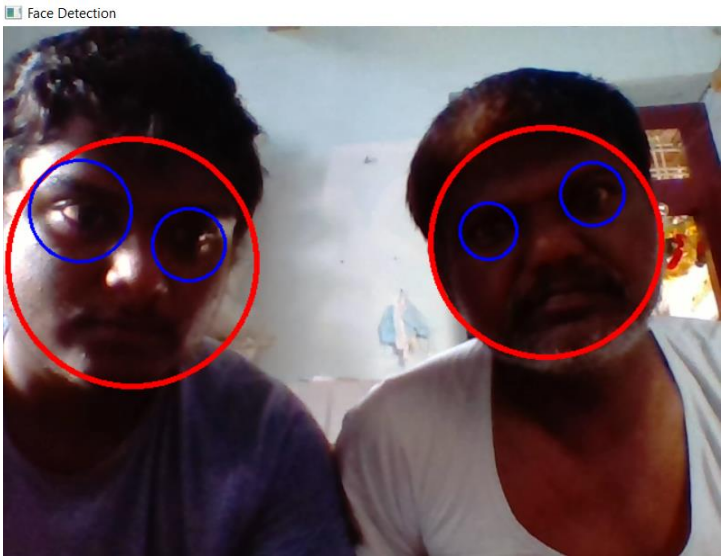
```
                        //circle(currimg, center, rad, eyeColor, 2, 8, 0);
                }

                /*Saves the cropped Current single face image with detected eyes*/
                resize(currimg, currimg, Size(), 4, 4);
                imwrite((finpath + "Face_" + to_string((i + 1)) + ".jpg"), currimg);
        }

        imshow("Face Detection", imgsrc); /*Shows all the faces and eyes spotted*/
        return;
}

int main()
{
        //string path = "0"; /*This is for WebCam. In case of video, stores the path*/
        VideoCapture cap(0); /* Tool to capture the Webcam. In case of Video, the argument passed
        will be path */
        Mat img;

        /*Path of Folder to store the different faces*/
        string savedpath = "Resources\\Faces\\";

        CascadeClassifier faceCascade, eyesCascade; /* PreDefined trained XML classifiers with facial
        features */

        /*Using haarcascades multilevel method --> Loading the classifiers*/
        faceCascade.load("Resources/haarcascade_frontalface_default.xml");
        eyesCascade.load("Resources/haarcascade_eye_tree_eyeglasses.xml");
        double scale = 1;

        if (cap.isOpened())
        {
                cout << "Press L to exit/leave the window" << endl;
                while (true)
                {
                        cap.read(img);
                        if (img.empty())
                                break;

                        detectImage(img, faceCascade, eyesCascade, scale, savedpath);

                        char c = (char)waitKey(30);
                        if (c == 'l' || c == 'L')   break;
                }
        }
        else
        {
                cout << "Cannot open Cam/ Video" << endl;
        }

        return 0;
        }
```

Output:

- On successful execution, all faces and eyes in real time are highlighted with the specified colours.
- Also, all the detected real time faces will be stored at the path specified in the code in .jpg format.

MY OUTPUT





Drawbacks:

- In case of background infiltration and poor quality, mostly incorrect results are obtained. This is because the Haarcascades XML classifier is a primitive method and hence needs high quality footage for accuracy.
- In a live stream or video, when the composure of an image changes, then the image will be detected as new image.
- Threshold area for filtering varies from video to video.
- Gives incorrect results when wearing spectacles.

By

Rekha Lokesh

19EC10052

E&ECE Department

IIT Kharagpur