University of Texas at Dallas

Erik Jonsson School of Engineering and Computer Science

OpenCV Cascade Classifiers

By: Mason Orsak

Course: CS 4v98.015

Semester: Spring 2021

Instructor: Dr. Richard Min

Assignment: Course Project

Student: Mason Orsak

Code Base: [GitHub](https://github.com/masonorsak/OpenCV_CS4v98.015)

Table of Contents

[Introduction 1](#_Toc68919745)

[Image Acquisition 2](#_Toc68919746)

[Face Image Set 2](#_Toc68919747)

[Tennis Ball Image Set 4](#_Toc68919748)

[Image Preparation 7](#_Toc68919749)

[Image Cataloging 8](#_Toc68919750)

[Haar Training 9](#_Toc68919751)

[Graphical User Interface 10](#_Toc68919752)

[Running the Project 14](#_Toc68919753)

# Introduction

This document includes technical details about this project, including code segments, pictures of the development process, and explanations for how to repeat this process. The entire code base is not included since it is rather extensive, however the complete code base can be found at the GitHub repository [here](https://github.com/masonorsak/OpenCV_CS4v98.015). While there were many sources of information used to develop this project (see project documentation) the two main ones were from python programming tutorials ([found here](https://pythonprogramming.net/loading-images-python-opencv-tutorial/)) and a research paper by the University of Auckland for training Haar cascades on Windows ([found here](https://www.cs.auckland.ac.nz/~m.rezaei/Tutorials/Creating_a_Cascade_of_Haar-Like_Classifiers_Step_by_Step.pdf)). For more information about the theory behind Haar cascade classifiers please direct your attention to the original paper that can be found [here](o%09https:/www.hpl.hp.com/techreports/Compaq-DEC/CRL-2001-1.pdf).

As described in the project documentation, there are five main parts to this project. Four of them for training the Haar cascades, and one for developing the graphical user interface. These five steps are image set acquisition, image set preparation, image cataloging, Haar cascade training, and GUI development. Each of these processes are discussed in detail in the following sections.

Id like to thank Dr. Richard Min for providing mentorship for this project and allowing me the opportunity to attend his undergraduate research position course. I have learned a lot about artificial intelligence, web programming, image processing, and graphical user interfaces. For any questions regarding this project feel free to contact me at [mto180000@utdallas.edu](mailto:mto180000@utdallas.edu).

# Image Acquisition

Here we gather the two sets of images that we use to train our cascade classifier. A set of positive images that contain the object we hope to identify, and a set of negative images that do not contain that object. If you find an online database of positive or negative images that will suffice for your project, simply download the image sets. Otherwise, we must resort to gather the images ourselves using sites like image-net. Image-net will give you a list of links that go directly to the image source for the object you requested. We will then use python to go link by link getting the photo, converting to greyscale, cropping it, scaling it, and saving the photo to one folder.

## Face Image Set

For our face Haar cascade positive image set I simply used the “Labeled Face in the Wild” database from the University of Massachusetts that can be found [here](http://vis-www.cs.umass.edu/lfw/). This database is exactly what it sounds like, a large database of various celebrity faces. Now all we must do is resize the images to 100 by 100 pixels and convert them all to black and white. This is easily done with the following python script.

import PIL

import os

import os.path

from PIL import Image, ImageOps

# Get directory of our faces in the wild download

f = r'C:/Users/Mason/Desktop/UTD Spring 2021/Undergrad Research - 4V98.015/Face Haar/tmp'

for file in os.listdir(f):  # for every image we read from this directory

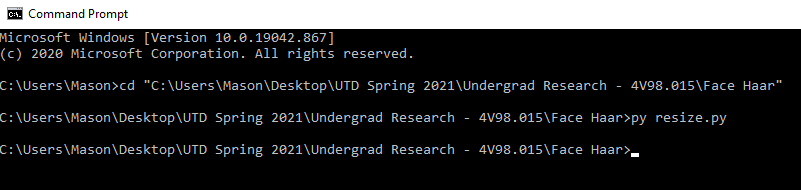
    f\_img = f+"/"+file              # get the path to the image

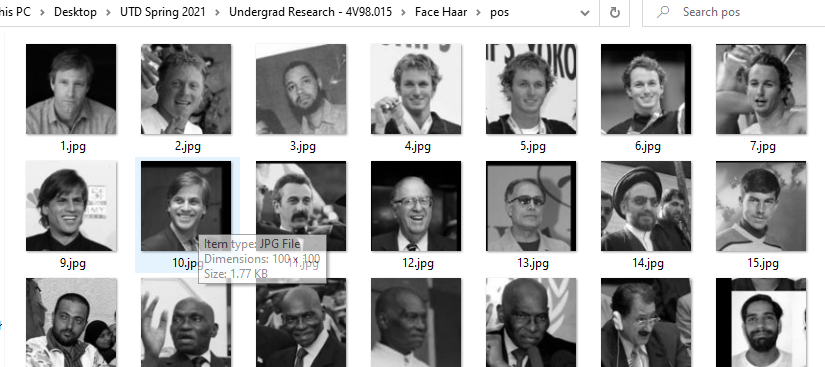
    img = Image.open(f\_img)         # opne the image

    img = ImageOps.grayscale(img)   # convert it to grayscale

    img = img.resize((100,100))     # resize it

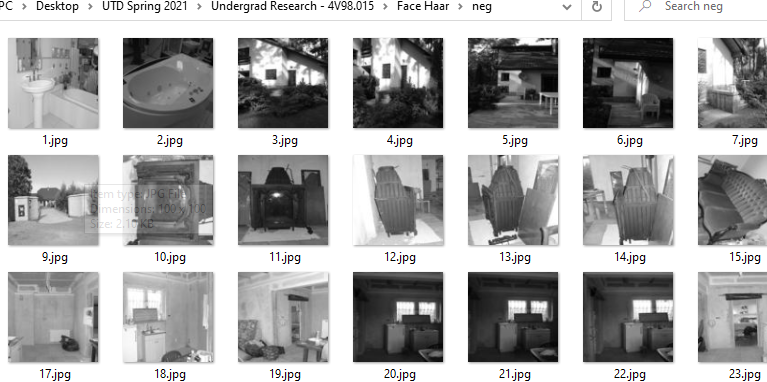
    img.save(f\_img)                 # and save it back to the directory





Now for the negative images we need any photo that does not have a face in it. We pull three different subsets of image from image net, links for each listed below. We then get every link to each image from image net, load the photo into OpenCV, resize it, convert it to black and white, then store it in our negative image directory.

* Neg 1: <http://image-net.org/api/text/imagenet.synset.geturls?wnid=n02913152>
* Neg 2: <http://image-net.org/api/text/imagenet.synset.geturls?wnid=n00017222>
* Neg 3: <http://image-net.org/api/text/imagenet.synset.geturls?wnid=n02084071>



## Tennis Ball Image Set

import urllib.request

import cv2

import numpy as np

import os

# takes image links from image-net and downloads the images

def store\_raw\_images():

    # where we are pulling image links from, get the links and decode

    neg\_images\_link = 'http://image-net.org/api/text/imagenet.synset.geturls?wnid=n02084071'

    neg\_image\_urls = urllib.request.urlopen(neg\_images\_link).read().decode()

    pic\_num = 1829  # how many pictures from this source we are using

    # for every image link we have donwload it

    for i in neg\_image\_urls.split('\n'):

        try:

            print(i) # what image we are on

            urllib.request.urlretrieve(i, "neg/"+str(pic\_num)+".jpg") # get the image and store it in the neg folder

            img = cv2.imread("neg/"+str(pic\_num)+".jpg",cv2.IMREAD\_GRAYSCALE) # load the image to opencv

            resized\_image = cv2.resize(img, (100, 100)) # resize the image

            cv2.imwrite("neg/"+str(pic\_num)+".jpg",resized\_image) # write the image to the directory

            pic\_num += 1

        except Exception as e:

            print(str(e))

store\_raw\_images()

Next, we repeat the exact same process we did pro the positive and negative image sets in face, but for a tennis ball. The only different is that for ball, both the positive and negative image sets were acquired from image-net. Listed below are the image links used for the tennis balls.

* Neg 1: <http://image-net.org/api/text/imagenet.synset.geturls?wnid=n00007846>
* Neg 2: <http://image-net.org/api/text/imagenet.synset.geturls?wnid=n02913152>
* Neg 3: <http://image-net.org/api/text/imagenet.synset.geturls?wnid=n00017222>
* Neg 4: <http://image-net.org/api/text/imagenet.synset.geturls?wnid=n02084071>
* Pos 1: <http://image-net.org/api/text/imagenet.synset.geturls?wnid=n04409515>



import urllib.request

import cv2

import numpy as np

import os

def store\_raw\_images():

    pos\_images\_link = 'http://image-net.org/api/text/imagenet.synset.geturls?wnid=n04409515'

    pos\_image\_urls = urllib.request.urlopen(pos\_images\_link).read().decode()

    pic\_num = 1

    for i in pos\_image\_urls.split('\n'):

        try:

            print(i)

            urllib.request.urlretrieve(i, "pos/"+str(pic\_num)+".jpg")

            img = cv2.imread("pos/"+str(pic\_num)+".jpg",cv2.IMREAD\_GRAYSCALE)

            # should be larger than samples / pos pic (so we can place our image on it)

            resized\_image = cv2.resize(img, (100, 100))

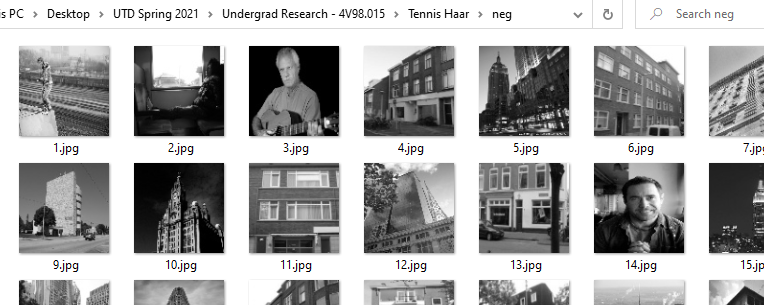
            cv2.imwrite("pos/"+str(pic\_num)+".jpg",resized\_image)

            pic\_num += 1

        except Exception as e:

            print(str(e))

store\_raw\_images()



import urllib.request

import cv2

import numpy as np

import os

def store\_raw\_images():

    neg\_images\_link = 'http://image-net.org/api/text/imagenet.synset.geturls?wnid=n02084071'

    neg\_image\_urls = urllib.request.urlopen(neg\_images\_link).read().decode()

    pic\_num = 1829

    for i in neg\_image\_urls.split('\n'):

        try:

            print(i)

            urllib.request.urlretrieve(i, "neg/"+str(pic\_num)+".jpg")

            img = cv2.imread("neg/"+str(pic\_num)+".jpg",cv2.IMREAD\_GRAYSCALE)

            # should be larger than samples / pos pic (so we can place our image on it)

            resized\_image = cv2.resize(img, (100, 100))

            cv2.imwrite("neg/"+str(pic\_num)+".jpg",resized\_image)

            pic\_num += 1

        except Exception as e:

            print(str(e))

store\_raw\_images()

# Image Preparation

Now that we have our two image sets, we must guarantee that no negative images are in our positive set, and vice-versa. This kind of cross contamination can wreak havoc on our final cascade classifier severely reducing its efficacy, so diligence in our image preparation is important.

We also need to remove images that did not load properly. Sometimes the images pulled from image-net no longer exist and instead blank placeholder images are downloaded. This will also reduce the efficiency of our final cascade classifier if we do not remove them. We do this using a python script that compares a failed to load image with every image in our positive and negative image sets.

import urllib.request

import cv2

import numpy as np

import os

def find\_uglies():

    match = False # have we found an image that failed to load (ugly image)

    for file\_type in ['neg']: # for every file in the negative image directory

        for img in os.listdir(file\_type): # get the image path

            for ugly in os.listdir('uglies'): # get the ugly image path

                try:

                    current\_image\_path = str(file\_type)+'/'+str(img) # full path of our current image

                    ugly = cv2.imread('uglies/'+str(ugly)) # hold our failed to load image

                    question = cv2.imread(current\_image\_path) # hold the image were currently comparing

                    if ugly.shape == question.shape and not(np.bitwise\_xor(ugly,question).any()): # if they match then delete the image

                        print('That is one ugly pic! Deleting!')

                        print(current\_image\_path)

                        os.remove(current\_image\_path)

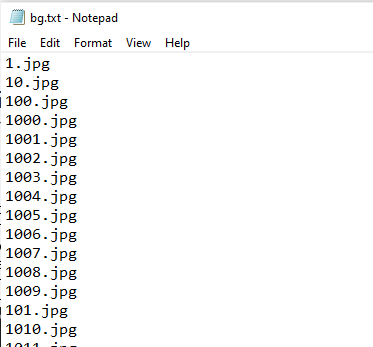
                except Exception as e:

                    print(str(e))

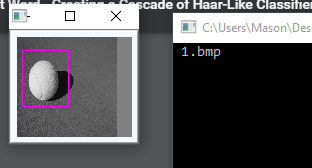
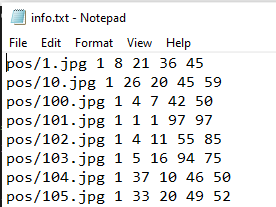
find\_uglies()

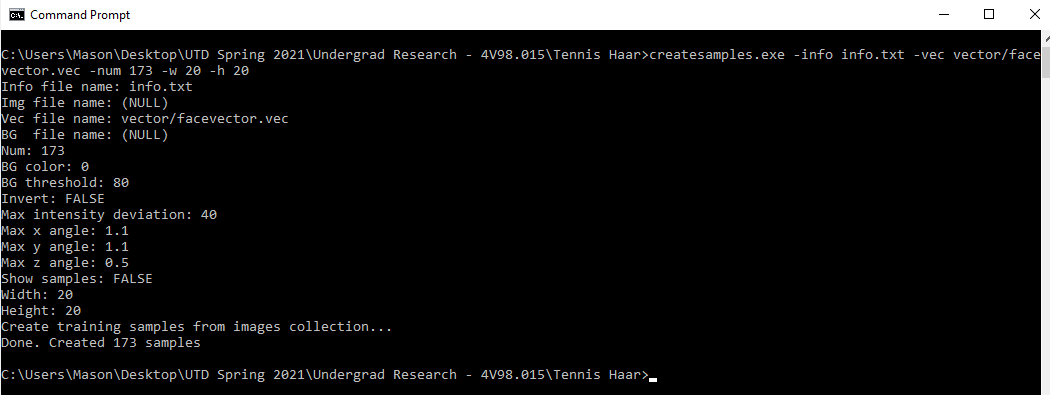
Example of the failed to load image.

# Image Cataloging

The OpenCV Haar training algorithm requires two text files as input. A negative text file that simply list the path to every image and its name, delimited by new lines. This is a very simple windows batch that just reads the file name of each image and writes it to a text file.

dir /b \*.jpg >bg.txt

Second, we need a positive image text file that holds the images path, name, how many objects are contained in the photo, and the coordinates of each of those objects (also new line delimited). This is one of the most time-consuming parts of creating Haar cascades. It can be done manually but is much faster with tools available online usually called “object markers for Haar cascades”, one such example can be found [here](https://github.com/dhruvvyas90/haar-object-marker). These tools provide graphical methods of scanning through each photo in a folder allowing the user to draw the region in which the object exists. The tools I used were provided in the University of Auckland paper and can be found [here](https://www.cs.auckland.ac.nz/~m.rezaei/Tutorials/Haar-Training.zip) labeled as “objectmaker.exe”.

Once we have the positive image text file with all the proper information about our images and where the objects lie within them, we convert it to a vector file. This takes our text file information and creates a list of vectors that our Haar cascade classifier will use for the machine learning algorithm. This is done with another simple windows batch file. The num flag is the number of positive pictures we have, the w and h flag is the width and height of our vectors. Take note of the width and height flag values, we will need these later.

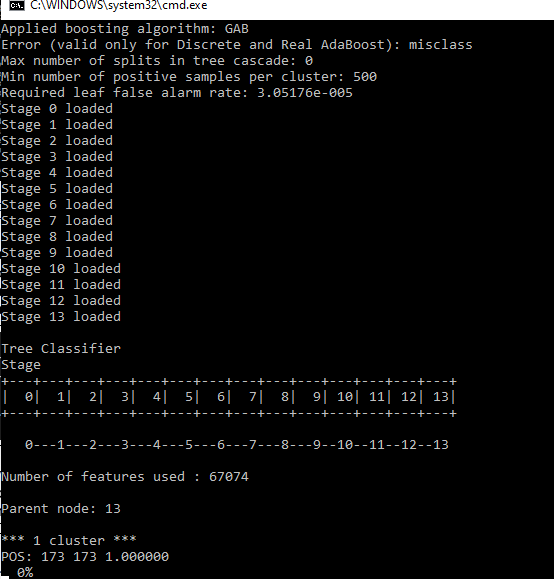
createsamples.exe -info info.txt -vec vector/facevector.vec -num 173 -w 20 -h 20

# Haar Training

Now that we have both of our image sets, the images are prepared and cataloged, we are finally ready to train our Haar cascade. This is done by running the haartraing.exe file that is provided in the University of Auckland tools. To run this though we need a few command line arguments. The most important of these is the width and height values, they MUST be the same as the values we defined above during create samples.

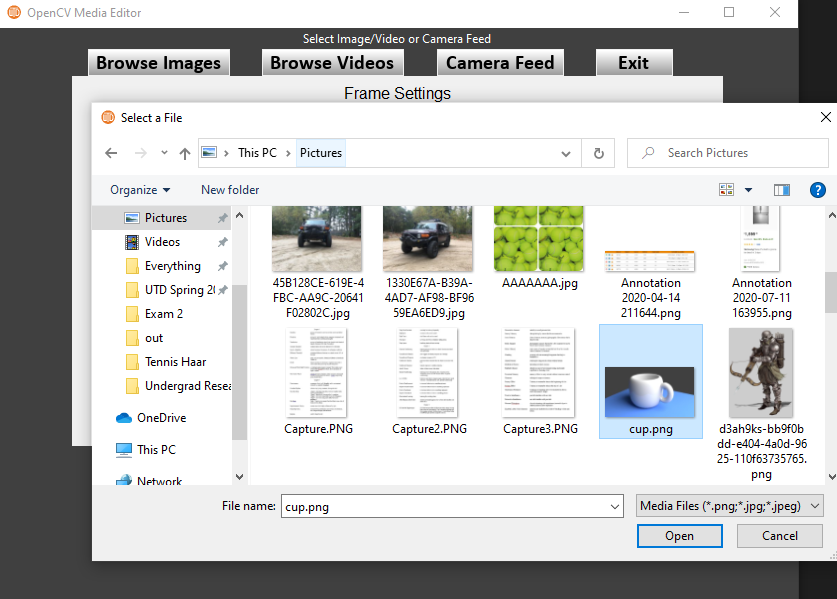
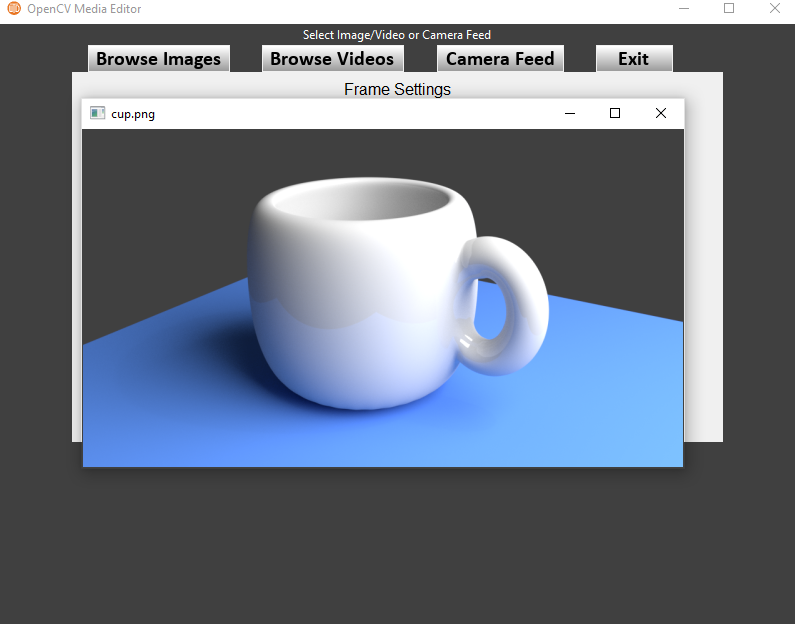
haartraining.exe -data cascades -vec vector/facevector.vec -bg neg/bg.txt -npos 173 -nneg 2000 -nstages 15 -mem 4096 -mode ALL -w 20 -h 20

rem -nonsym

Depending on many factors, like the size of your images, how many images you have, your computers RAM, etc. The training itself will likely take a few hours at least, so be prepared to run it in the background for a while. Luckily for whatever reason if you need to stop the training, you can run the training command again and the algorithm will pick up where it left off. Once the training is complete, we will receive a xml file that contains our finished Haar cascade!

# Graphical User Interface

The graphical user interface code is quite long since most of it is graphics declaration, so for the sake of length of this document I will only include excerpts from it. The full GUI code can be found on the GitHub repository [here](https://github.com/masonorsak/OpenCV_CS4v98.015).

This GUI will contain three sources of input, images, videos, and a live camera feed. The user will be able to select which one of these inputs they would like to operate on. If image or video input options are selected, a dialog box will open prompting them to select what video or image they want to choose. Only png, jpg, and jpeg image types are allowed since those are the only types OpenCV can operate on. The video player allows video files of type mov, mp4, and avi.

One every selected input the program will look for both faces and tennis balls within each frame or image it reads. If one is detected it will outline it with a box and label the area with the type of object it found.

def Detect(self, frame, use\_pro):

        # Ensure the image has 3 color channels or else cvtColor will crash app

        if len(frame.shape) > 2:

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

        else:

            gray = frame

        # if were using a pro cascade then do that, otherwise use our cascade

        if (use\_pro == True):

            faces = self.pro\_face\_cascade.detectMultiScale(gray, 1.3, 5)

        else:

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

        tennis = self.tennis\_cascade.detectMultiScale(gray, 1.3, 5)

        # the cascade returns x and y coord of where the haar detected the face

        # and the width and height of that face

        for (x,y,w,h) in faces:

            # change the color of the detected object based on our selected RGB values

            frame = self.faceColor.ChannelEdit(frame, (x,y,w,h))

            # draw a rectangle around our detected face objects

            cv2.rectangle(frame,(x,y),(x+w,y+h),(0,255,255),2)

            cv2.putText(frame, 'Face', (x, y-10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 255, 255), 2, cv2.LINE\_AA)

        # the cascade returns x and y coord of where the haar detected the ball

        # and the width and height of that ball

        for (x,y,w,h) in tennis:

            # change the color of the detected object based on our selected RGB values

            frame = self.ballColor.ChannelEdit(frame, (x,y,w,h))

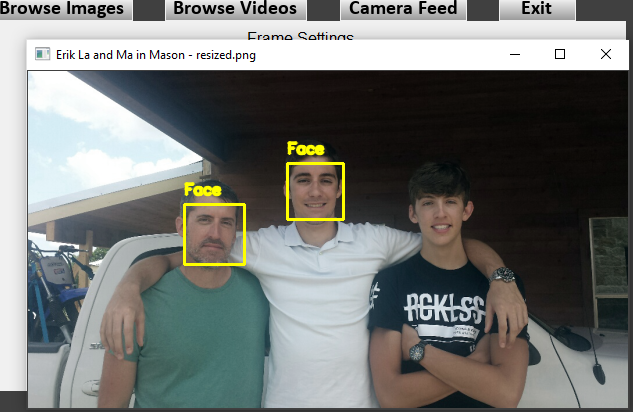
            # draw a rectangle around our detected ball objects

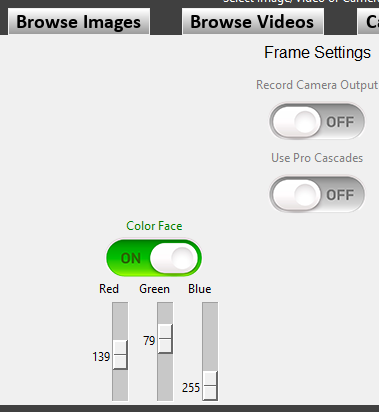
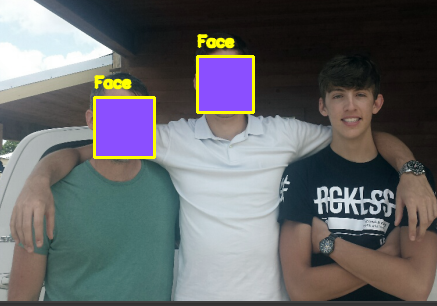
            cv2.rectangle(frame,(x,y),(x+w,y+h),(255,255,0),2)

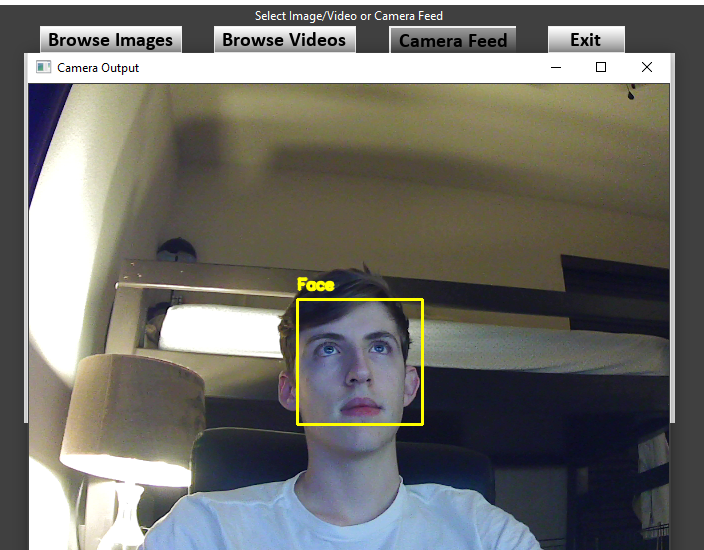
            cv2.putText(frame, 'Ball', (x, y-10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (255, 255, 0), 2, cv2.LINE\_AA)

        # return the detected object so the players can display it

        return frame



I have learned in my testing that my face Haar cascade really does not do well with faces that have bangs like mine. It seems not many of the faces in my positive image set had bangs. Anyway, the user also has the option to specify a color in terms of red, green, and blue values from 0 to 255. This will change the color of the detected objects to what the user has specified, like so.

The GUI also allows the functionality to record live camera feed data (if applicable given the input), you simply need to click the “Record Camera Output” button before starting the camera. The file will be saved as “output.avi” in the directory in which the gui.py file is located. Using a professional cascade classifier instead of the ones I have created is also an option I have included so you can see the difference larger image sets can make. The professional cascade classifier is obtained from OpenCV’s GitHub that can be found [here](https://github.com/opencv/opencv/tree/master/data/haarcascades).

# Running the Project

To run this project, download the code base from the GitHub repository [here](https://github.com/masonorsak/OpenCV_CS4v98.015). The only files/folders needed to run the finished project is images, FaceHaarCascade.xml, ProFaceHaarCascade.xml, TennisBallHaarCascade.xml, and gui.py. All other files are either for documentation or training the Haar cascades.

Open the windows command prompt by pressing “widows-key + S” then typing “command prompt”. Navigate to the directory you downloaded the unzipped code repository to using like so “cd C:\Users\Mason\Desktop\OpenCV\_CS4v98.015-master\”. Then execute the GUI by typing “py gui.py”. NOTE: you must have python 3 and all libraries listed at the top of gui.py installed.

