Individual Final Report
Cristina Giraldo

Introduction

At a basic level, facial recognition works by obtaining geometry by scanning the face and recognizing patterns such as the distance between the eyes, the size of the nose and mouth, and so on. With that information, the computer can create a virtual map of the face and is then able to perform a match against other faces to identify the appearance of the person who is being captured through a camera or a digital image (Bala & Watney, 2019).

Nowadays, facial recognition has become a subject of great importance in various fields. For example, it is used in law enforcement to identify criminals, in social media to recognize friends, in transportation to drive cars without a driver, in smart devices to unlock the device with just the look of the face and in surveillance to control communities. Given all these fascinating applications, we are interested to understand how this technology works by using facial recognition in real time and implementing web scraping to obtain basic information about Instagram users' accounts and to present such information on the facial recognition.

Description of individual work

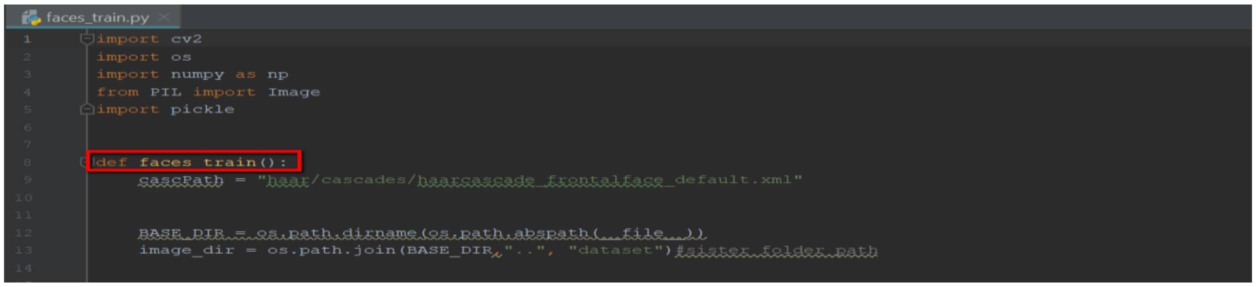
At the beginning we were doing research about how to implement facial recognition. After we got the idea. We started to implement it. I particularly was working on the HOG while my classmate was working on the algorithm HAAR.

The next step was to improve our project and we started learning about web scraping and how it would be possible to link Instagram's user information with the process of facial recognition.

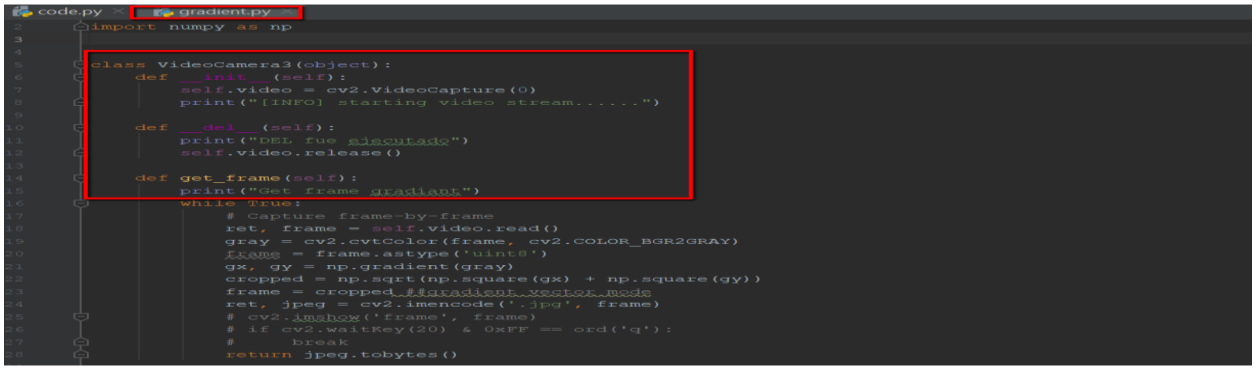
Once we had all the algorithms working, I particularly focused on the UI by using Dash Plotly. Once the layout was ready, I implemented some classes to put to work the project all together. However, it is worth clarifying that this was a team effort and at the end we both participated in each file code given that sometimes we need to fix an issue or sometimes we wanted to implement a new functionality.

Description of Work / Code

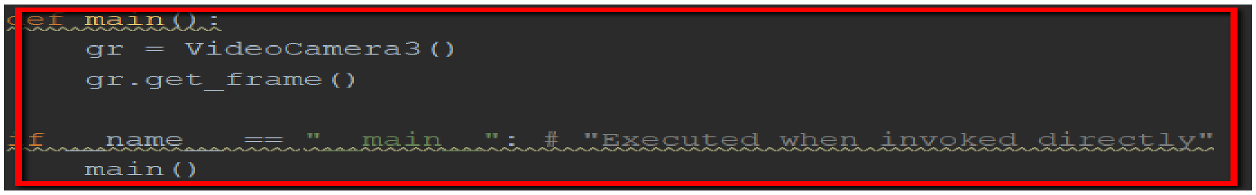
In this project, I had an active participation in the creation of classes, methods and UI. To be more precise under the folder Individual-Final-Project-Report/ Individual-Final-Project-Report-Cristina-Giraldo/Code the coding done by me can be found. In that is demonstrated my active participation in the team during the process of construction of the algorithm implemented for the facial recognition and web scraping in real time.



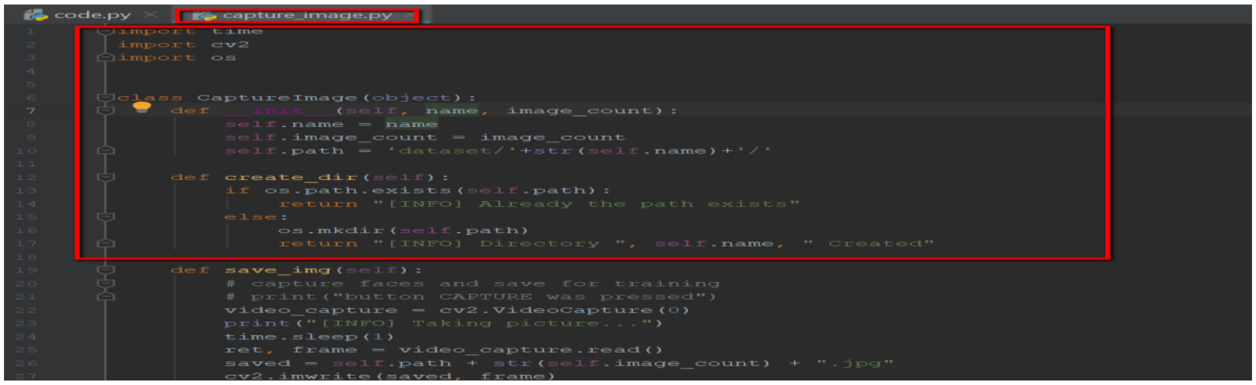
```python
import cv2
import os
import numpy as np
from PIL import Image
import pickle


def faces_train():
    cascPath = "haar/cascades/haarcascade_frontalface_default.xml"


    BASE_DIR = os.path.dirname(os.path.abspath(__file__))
    image_dir = os.path.join(BASE_DIR,"..", "dataset")#sister folder path

```



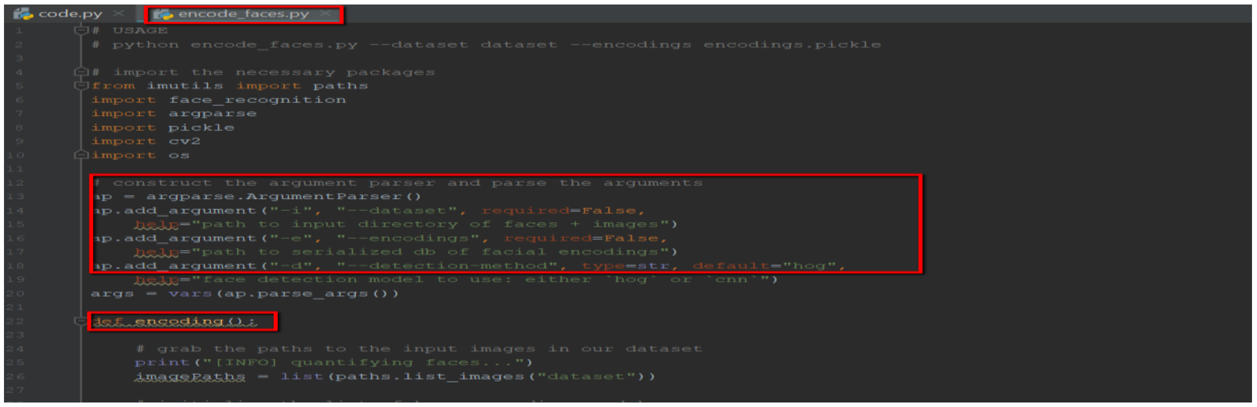
```python
import numpy as np


class VideoCamera3(object):
    def __init__(self):
        self.video = cv2.VideoCapture(0)
        print("[INFO] starting video stream......")

    def __del__(self):
        print("DEL fue ejecutado")
        self.video.release()

    def get_frame(self):
        print("Get frame gradiant")
        while True:
            # Capture frame-by-frame
            ret, frame = self.video.read()
            gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
            frame = frame.astype('uint8')
            gx, gy = np.gradient(gray)
            cropped = np.sqrt(np.square(gx) + np.square(gy))
            frame = cropped #gradient vector mode
            ret, jpeg = cv2.imencode('.jpg', frame)
            # cv2.imshow('frame', frame)
            # if cv2.waitKey(20) & 0xFF == ord('q'):
            #     break
            return jpeg.tobytes()
```



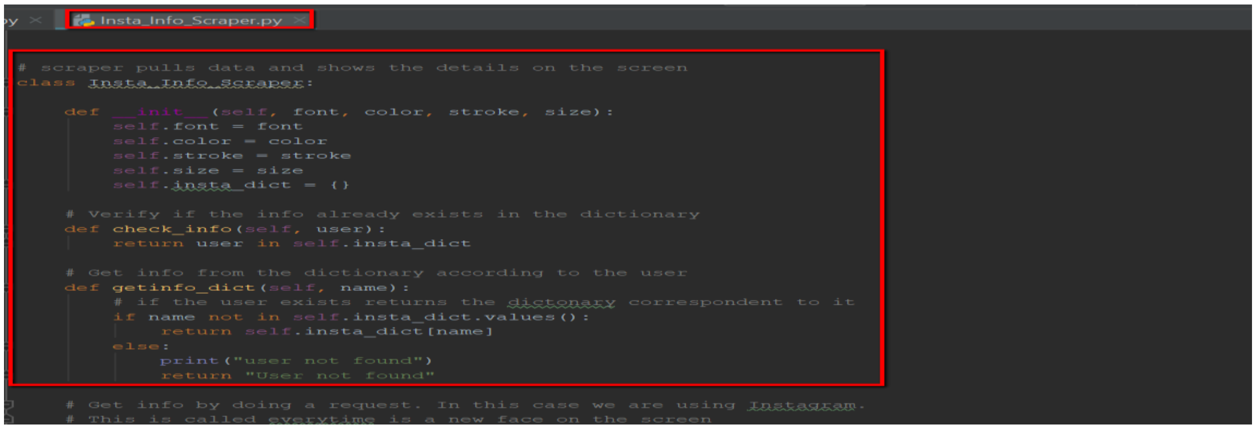
```python
def main():
    gr = VideoCamera3()
    gr.get_frame()

if __name__ == "__main__": # "Executed when invoked directly"
    main()
```



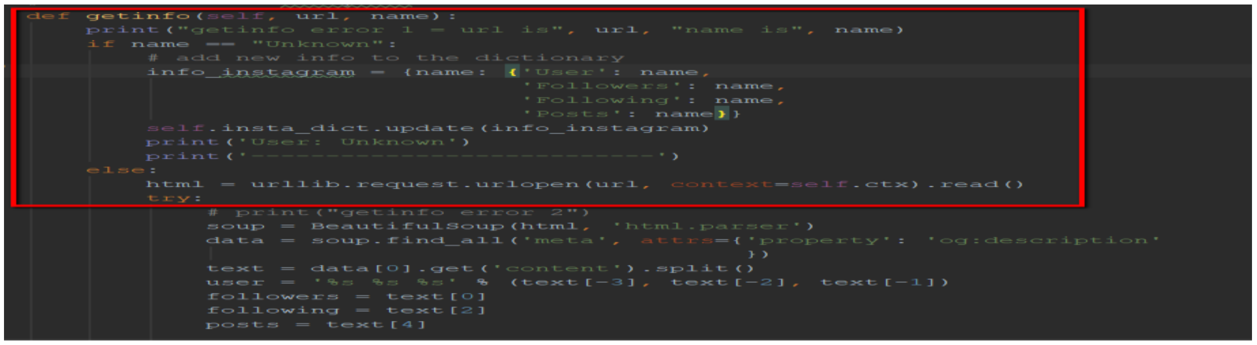
```python
import time
import cv2
import os


class CaptureImage(object):
    def __init__(self, name, image_count):
        self.name = name
        self.image_count = image_count
        self.path = 'dataset/'+str(self.name)+'/'

    def create_dir(self):
        if os.path.exists(self.path):
            return "[INFO] Already the path exists"
        else:
            os.mkdir(self.path)
            return "[INFO] Directory ", self.name, " Created"

    def save_img(self):
        # capture faces and save for training
        # print("button CAPTURE was pressed")
        video_capture = cv2.VideoCapture(0)
        print("[INFO] Taking picture...")
        time.sleep(1)
        ret, frame = video_capture.read()
        saved = self.path + str(self.image_count) + ".jpg"
        cv2.imwrite(saved, frame)
```



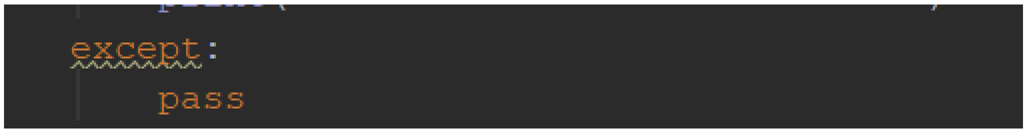
```python
# USAGE
# python encode_faces.py --dataset dataset --encodings encodings.pickle

# import the necessary packages
from imutils import paths
import face_recognition
import argparse
import pickle
import cv2
import os

# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--dataset", required=False,
    help="path to input directory of faces + images")
ap.add_argument("-e", "--encodings", required=False,
    help="path to serialized db of facial encodings")
ap.add_argument("-d", "--detection-method", type=str, default="hog",
    help="face detection model to use: either `hog` or `cnn`")
args = vars(ap.parse_args())

def encoding():

    # grab the paths to the input images in our dataset
    print("[INFO] quantifying faces...")
    imagePaths = list(paths.list_images("dataset"))
```



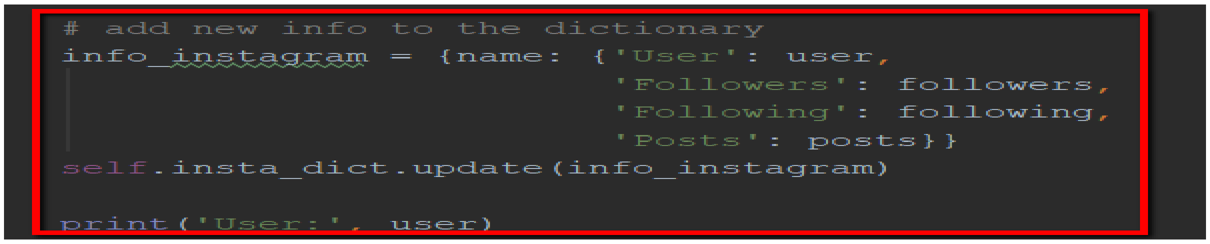
```python
# scraper pulls data and shows the details on the screen
class Insta_Info_Scraper:

    def __init__(self, font, color, stroke, size):
        self.font = font
        self.color = color
        self.stroke = stroke
        self.size = size
        self.insta_dict = {}

    # Verify if the info already exists in the dictionary
    def check_info(self, user):
        return user in self.insta_dict

    # Get info from the dictionary according to the user
    def getinfo_dict(self, name):
        # if the user exists returns the dictonary correspondent to it
        if name not in self.insta_dict.values():
            return self.insta_dict[name]
        else:
            print("user not found")
            return "User not found"

    # Get info by doing a request. In this case we are using Instagram.
    # This is called everytime is a new face on the screen
```



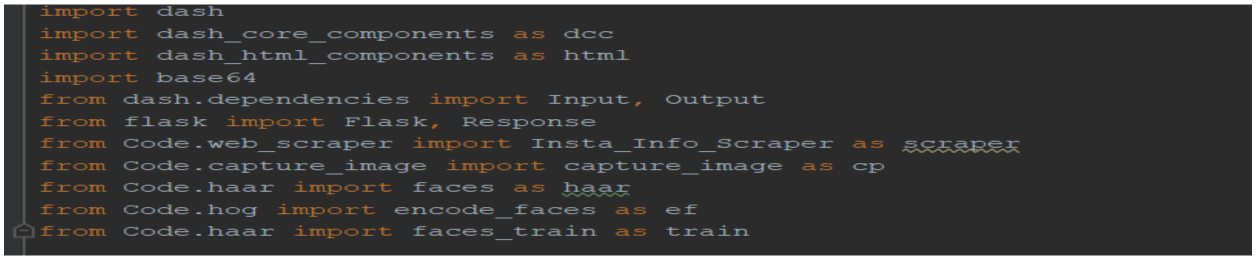
```python
def getinfo(self, url, name):
    print("getinfo error 1 - url is", url, "name is", name)
    if name == "Unknown":
        # add new info to the dictionary
        info_instagram = {name: {'User': name,
                                 'Followers': name,
                                 'Following': name,
                                 'Posts': name}}
        self.insta_dict.update(info_instagram)
        print('User: Unknown')
        print('------------------------------')
    else:
        html = urllib.request.urlopen(url, context=self.ctx).read()
        try:
            # print("getinfo error 2")
            soup = BeautifulSoup(html, 'html.parser')
            data = soup.find_all('meta', attrs={'property': 'og:description'
                                                })
            text = data[0].get('content').split()
            user = '%s %s %s' % (text[-3], text[-2], text[-1])
            followers = text[0]
            following = text[2]
            posts = text[4]
```



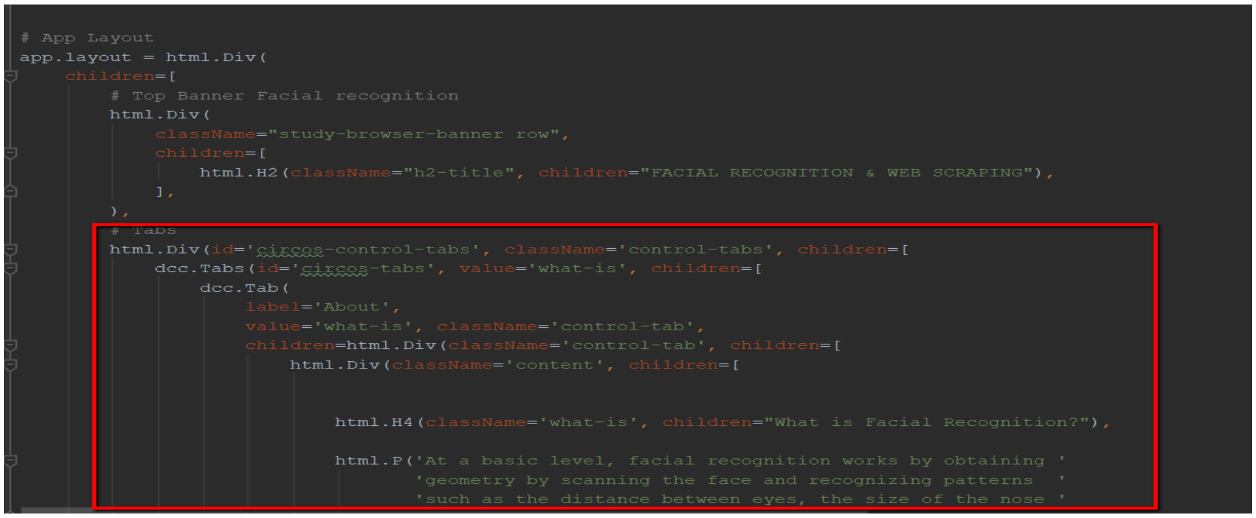
```python
except:
    pass
```



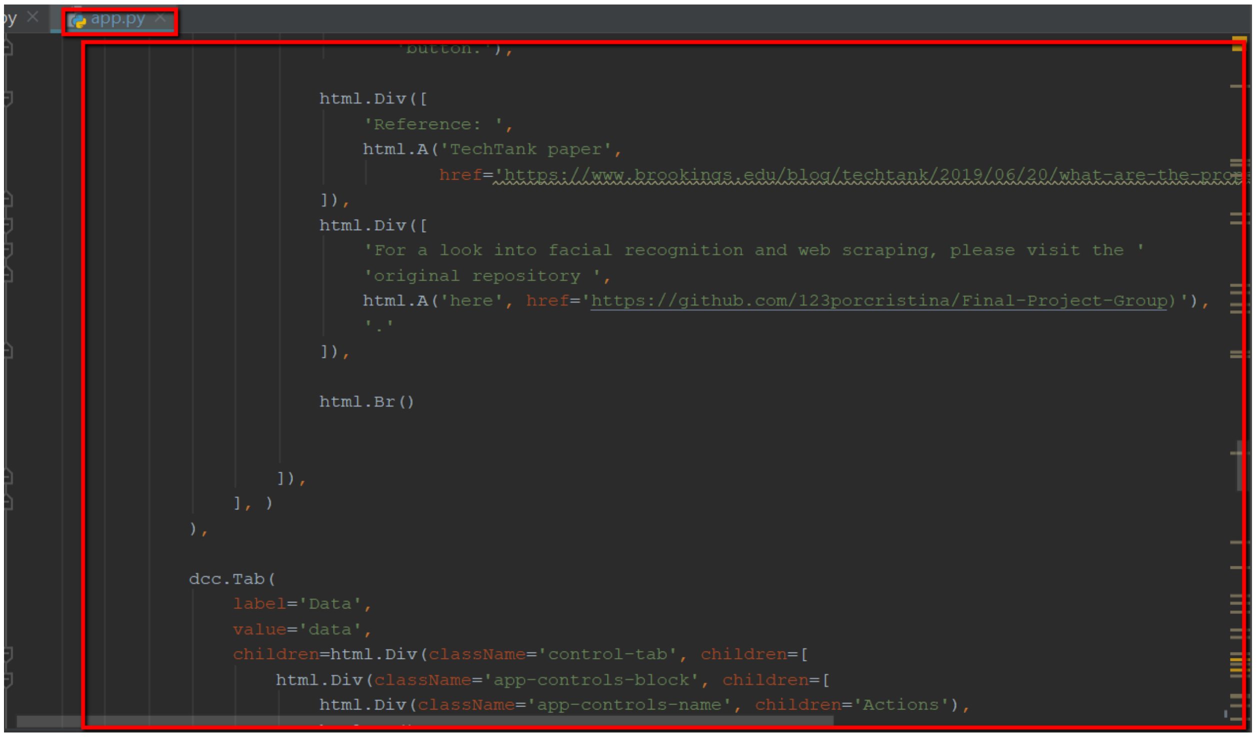
```
# add new info to the dictionary
info_instagram = {name: {'User': user,
                        'Followers': followers,
                        'Following': following,
                        'Posts': posts}}
self.insta_dict.update(info_instagram)

print('User:', user)
```



```
import dash
import dash_core_components as dcc
import dash_html_components as html
import base64
from dash.dependencies import Input, Output
from flask import Flask, Response
from Code.web_scraper import Insta_Info_Scraper as scraper
from Code.capture_image import capture_image as cp
from Code.haar import faces as haar
from Code.hog import encode_faces as ef
from Code.haar import faces_train as train
```

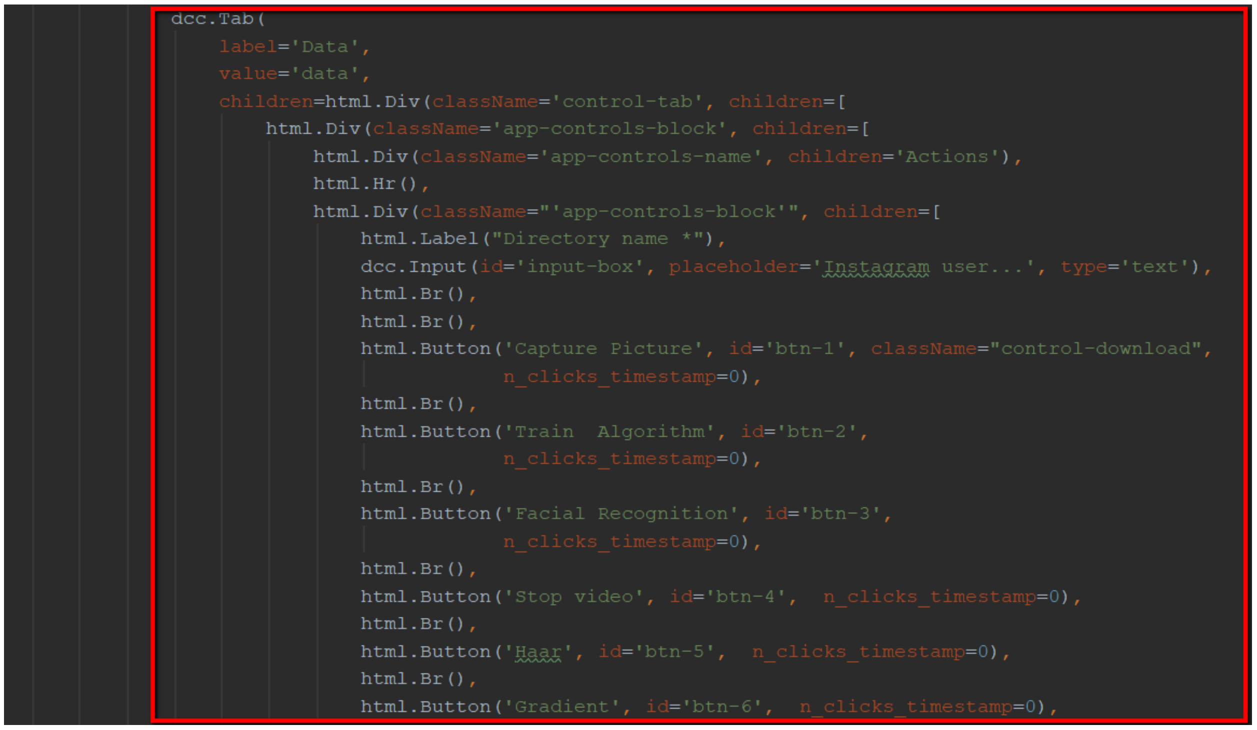


```
# App Layout
app.layout = html.Div(
    children=[
        # Top Banner Facial recognition
        html.Div(
            className="study-browser-banner row",
            children=[
                html.H2(className="h2-title", children="FACIAL RECOGNITION & WEB SCRAPING"),
            ],
        ),
        # Tabs
        html.Div(id='circos-control-tabs', className='control-tabs', children=[
            dcc.Tabs(id='circos-tabs', value='what-is', children=[
                dcc.Tab(
                    label='About',
                    value='what-is', className='control-tab',
                    children=html.Div(className='control-tab', children=[
                        html.Div(className='content', children=[


                            html.H4(className='what-is', children="What is Facial Recognition?"),

                            html.P('At a basic level, facial recognition works by obtaining '
                                   'geometry by scanning the face and recognizing patterns  '
                                   'such as the distance between eyes, the size of the nose '
```

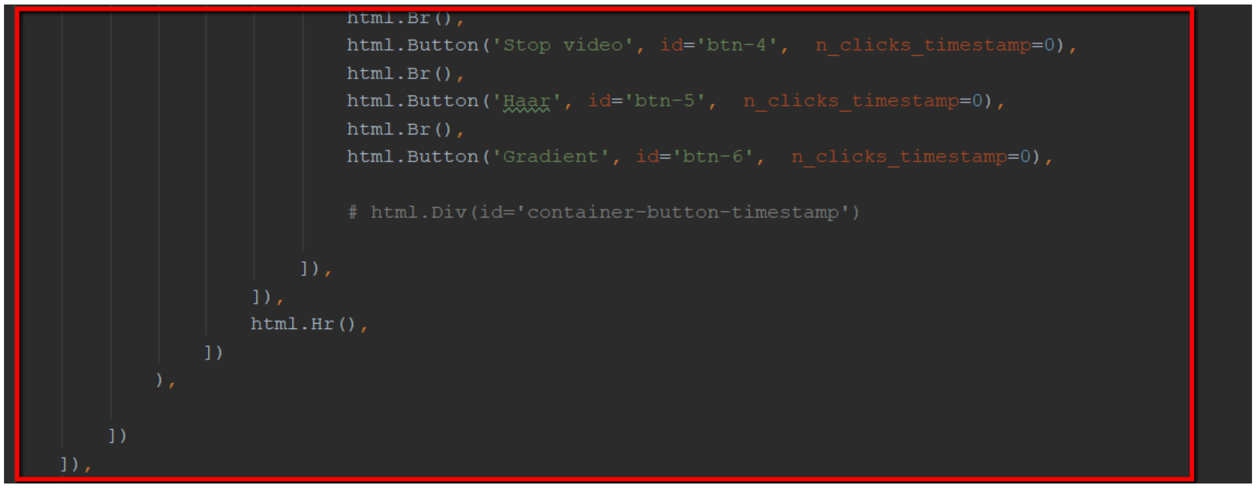


app.py

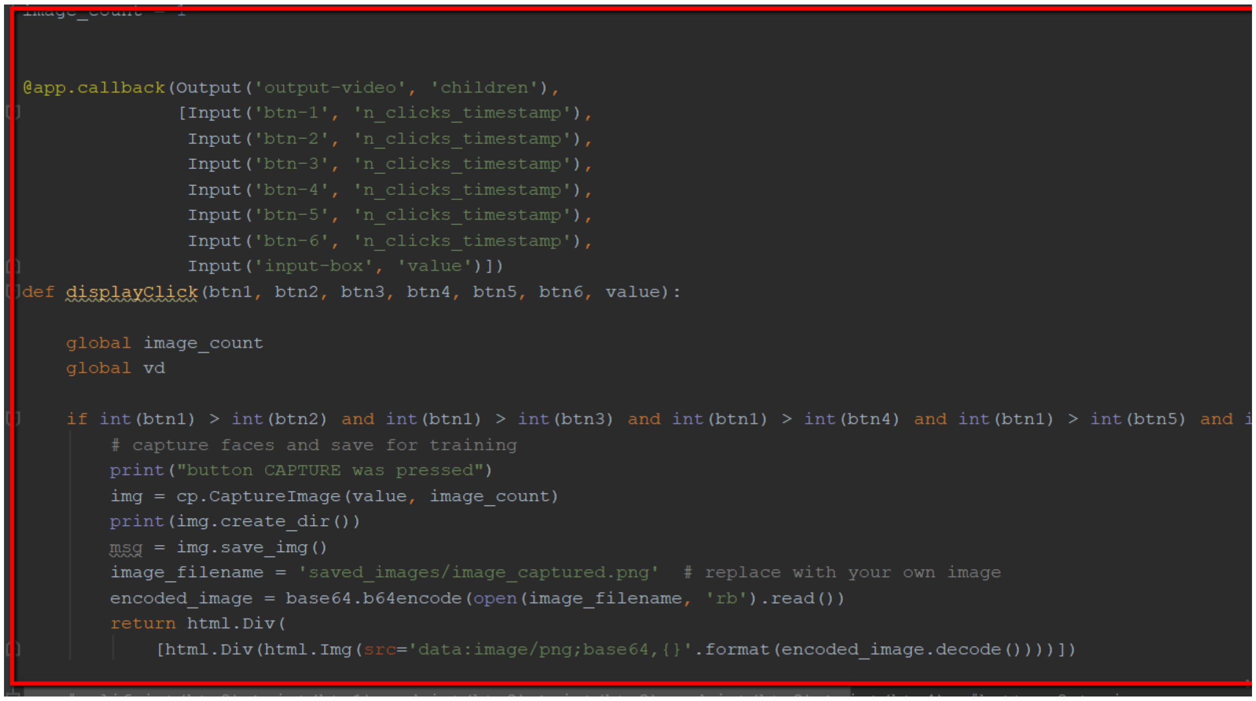
```
                    'button.'),

            html.Div([
                'Reference: ',
                html.A('TechTank paper',
                       href='https://www.brookings.edu/blog/techtank/2019/06/20/what-are-the-pro
            ]),
            html.Div([
                'For a look into facial recognition and web scraping, please visit the '
                'original repository ',
                html.A('here', href='https://github.com/123porcristina/Final-Project-Group)'),
                '.'
            ]),

            html.Br()


        ]),
    ], )
),

dcc.Tab(
    label='Data',
    value='data',
    children=html.Div(className='control-tab', children=[
        html.Div(className='app-controls-block', children=[
            html.Div(className='app-controls-name', children='Actions'),
```



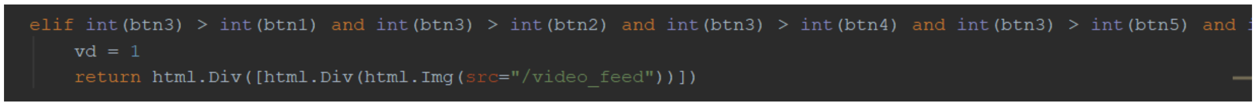
```
dcc.Tab(
    label='Data',
    value='data',
    children=html.Div(className='control-tab', children=[
        html.Div(className='app-controls-block', children=[
            html.Div(className='app-controls-name', children='Actions'),
            html.Hr(),
            html.Div(className="'app-controls-block'", children=[
                html.Label("Directory name *"),
                dcc.Input(id='input-box', placeholder='Instagram user...', type='text'),
                html.Br(),
                html.Br(),
                html.Button('Capture Picture', id='btn-1', className="control-download",
                            n_clicks_timestamp=0),
                html.Br(),
                html.Button('Train  Algorithm', id='btn-2',
                            n_clicks_timestamp=0),
                html.Br(),
                html.Button('Facial Recognition', id='btn-3',
                            n_clicks_timestamp=0),
                html.Br(),
                html.Button('Stop video', id='btn-4',  n_clicks_timestamp=0),
                html.Br(),
                html.Button('Haar', id='btn-5',  n_clicks_timestamp=0),
                html.Br(),
                html.Button('Gradient', id='btn-6',  n_clicks_timestamp=0),
```



```
                                    html.Br(),
                                    html.Button('Stop video', id='btn-4',  n_clicks_timestamp=0),
                                    html.Br(),
                                    html.Button('Haar', id='btn-5',  n_clicks_timestamp=0),
                                    html.Br(),
                                    html.Button('Gradient', id='btn-6',  n_clicks_timestamp=0),

                                    # html.Div(id='container-button-timestamp')

                                ]),
                            ]),
                            html.Hr(),
                        ])
                    ),

                ])
            ]),
```



```
image_count = 1


@app.callback(Output('output-video', 'children'),
              [Input('btn-1', 'n_clicks_timestamp'),
               Input('btn-2', 'n_clicks_timestamp'),
               Input('btn-3', 'n_clicks_timestamp'),
               Input('btn-4', 'n_clicks_timestamp'),
               Input('btn-5', 'n_clicks_timestamp'),
               Input('btn-6', 'n_clicks_timestamp'),
               Input('input-box', 'value')])
def displayClick(btn1, btn2, btn3, btn4, btn5, btn6, value):

    global image_count
    global vd

    if int(btn1) > int(btn2) and int(btn1) > int(btn3) and int(btn1) > int(btn4) and int(btn1) > int(btn5) and i
        # capture faces and save for training
        print("button CAPTURE was pressed")
        img = cp.CaptureImage(value, image_count)
        print(img.create_dir())
        msg = img.save_img()
        image_filename = 'saved_images/image_captured.png'  # replace with your own image
        encoded_image = base64.b64encode(open(image_filename, 'rb').read())
        return html.Div(
            [html.Div(html.Img(src='data:image/png;base64,{}'.format(encoded_image.decode())))])
```



```
elif int(btn3) > int(btn1) and int(btn3) > int(btn2) and int(btn3) > int(btn4) and int(btn3) > int(btn5) and i
    vd = 1
    return html.Div([html.Div(html.Img(src="/video_feed"))])
```

Results

For the Haar algorithm about 23 pictures were trained. Most of the images belonged to Gregg. After the algorithm was executed, we noticed that the accuracy was not precise (the result can be seen on figure 1) and the algorithm worked only under certain conditions, for example just for frontal face. However, Haar provided a good speed on the video and the recognition.

For the HOG algorithm fewer digital images were necessary to train, and the facial recognition worked well under different conditions and different positions (see figure 2). Thus, the HOG method could predict an individual correctly in about 80% of the time while the Haar algorithm had gave us a confidence of about 35% due to very limited number of images used.

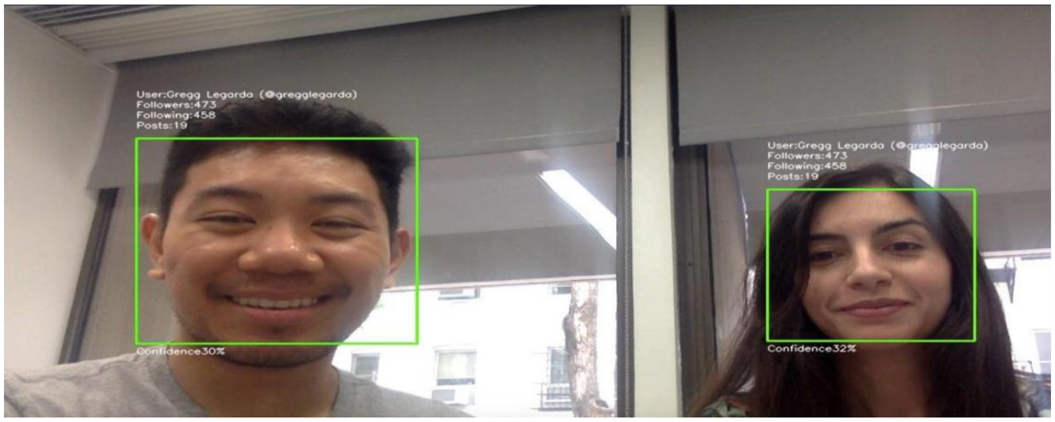


Figure 1. Facial recognition using Haar-like feature based

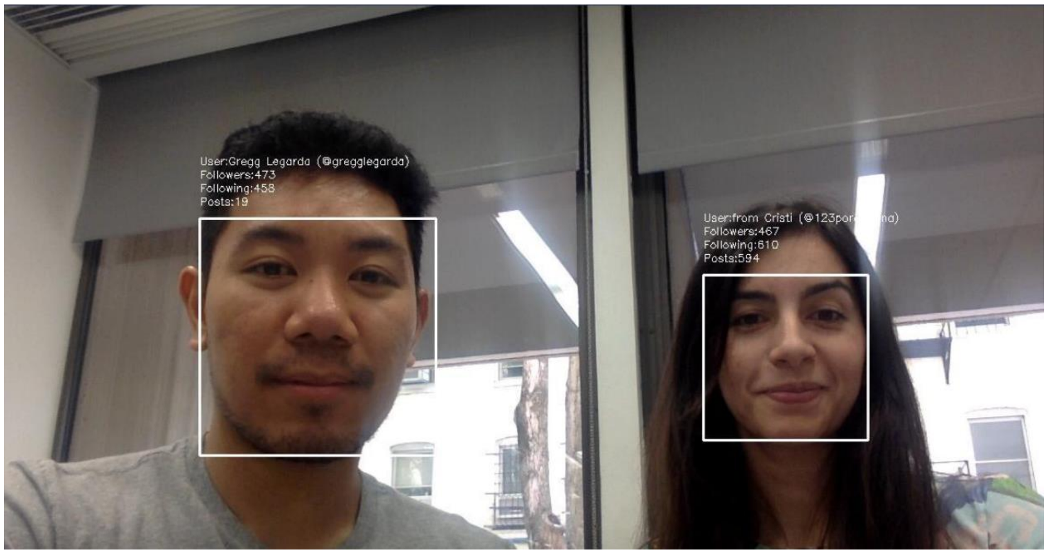


Figure 2. Facial recognition using Histogram of Gradients (HOG)

Finally, All the algorithms were integrated in a web application for which I used dash plotly. The result is an appealing application, user friendly and very easy to use (see figure 3).

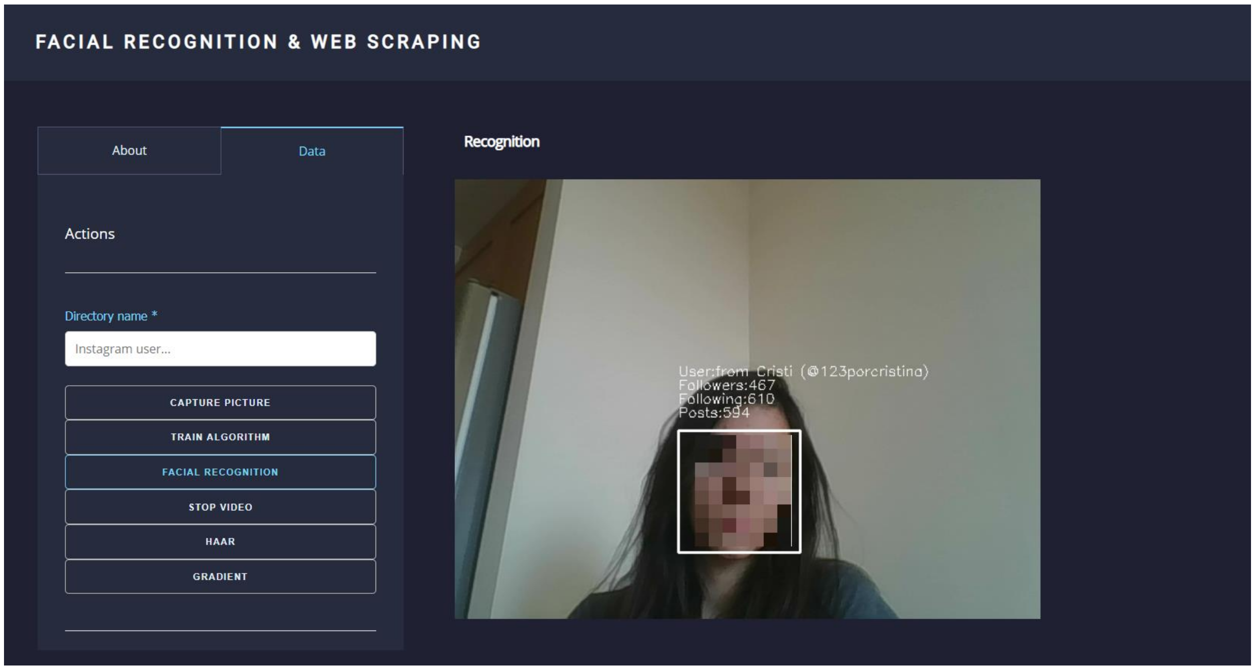


Figure 3. User interface

Summary and conclusions

Our project used two different techniques to perform facial recognition along with web scraping to obtain information related to the user that appears on the screen. These techniques allowed us to implemented algorithms in different situations and learn how each of the algorithms worked. In addition, we have a clear idea when it would be useful to use each of the techniques. Although, both algorithms work well, each implementation have disadvantages most of the times related to the speed. For instance, HOG algorithm is significant slower than Haar when the facial recognition is performed in real time. On the other hand, Haar algorithm tend to be slower for training but also less accurate.

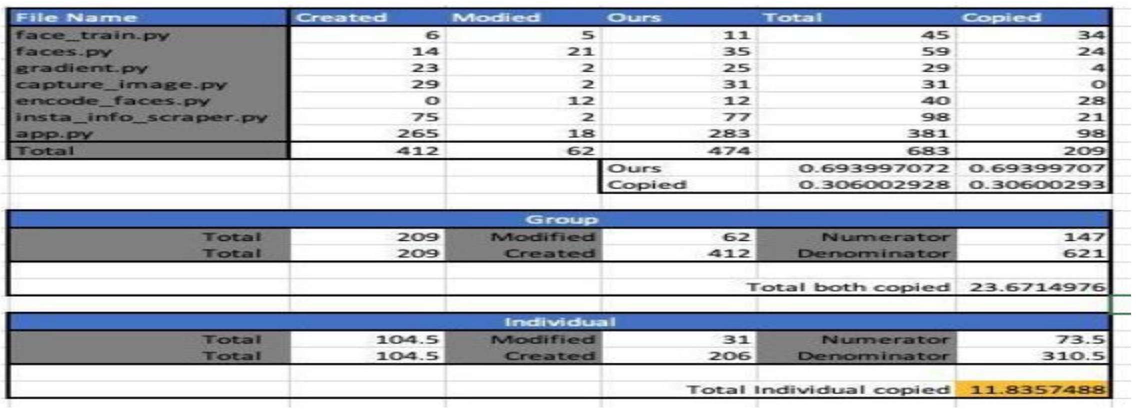
To improve the accuracy, I would propose to implement this code along with Supporting Vector Machine given that this technique is particularly good at image classification. However, if the algorithm is still not accurate, it would be a good idea to implement an additional algorithm to improve the process of classification. Another improvement that can be done is to train the algorithm with oriental faces and save the training to be use as a pretraining in the algorithms. This solution would help to reduce the misclassification performed in this type of images.

According to several paper that I read, it is mentioned that YOLO (You Only Look Once) is another algorithm that can be used. Yolo is highly accurate, but it is computational expensive. However, several people argued that implementing a convolutional neural network would be a better approach due to CNN is easier to use and it is less computational expensive. However, is worth to mention that the face recognition library works with dlib. Dlib library helps to perform the facial recognition by implementing a ResNet network with 29 convolutional layers. In addition, the pre-trained model used in face recognition library was done by training three million of images (King, 2017). Although, facial recognition library provides the possibility to perform a CNN for the facial recognition as well, in this project we decided to use HOG given that our computers did not have the power to process a convolutional neural network. Even though, by using Histogram of Gradients we got good results at the process of recognizing the person on the screen

Finally, it is worth to mention that we all participated in different steps of the project and I particularly feel proud of the final project. We learned about different types of facial recognition, histogram of gradients, gradients and some about CNN (due to all the reading that I read.). I learn how the different algorithms for facial recognition work and how they are implemented. Also, we learn about the limitations, disadvantages and advantages. This project made me even get more curious about computing vision.

Percentage of code

11.83%



File Name	Created	Modied	Ours	Total	Copied
face_train.py	6	5	11	45	34
faces.py	14	21	35	59	24
gradient.py	23	2	25	29	4
capture_image.py	29	2	31	31	0
encode_faces.py	0	12	12	40	28
insta_info_scraper.py	75	2	77	98	21
app.py	265	18	283	381	98
Total	412	62	474	683	209
			Ours	0.693997072	0.69399707
			Copied	0.306002928	0.30600293

Group					
Total	209	Modified	62	Numerator	147
Total	209	Created	412	Denominator	621
				Total both copied	23.6714976

Individual					
Total	104.5	Modified	31	Numerator	73.5
Total	104.5	Created	206	Denominator	310.5
				Total Individual copied	11.8357488

References

Bala, N., & Watney, C. (2019, June 19). What are the proper limits on police use of facial recognition? Retrieved from https://www.brookings.edu/blog/techtank/2019/06/20/what-are-the-proper-limits-on-police-use-of-facial-recognition/

King, D. (2017, February 12). High Quality Face Recognition with Deep Metric Learning. Retrieved from http://blog.dlib.net/2017/02/high-quality-face-recognition-with-deep.html