Department of Computer Science
Technical University of Cluj-Napoca

Structure of Computer Systems

Laboratory activity 2019-2020

Project title: Analysis program of the face's morphological features

Name: Bindea Bogdan Nicușor
Group: 30433
Email: Bogdan.Bindea@student.utcluj.ro

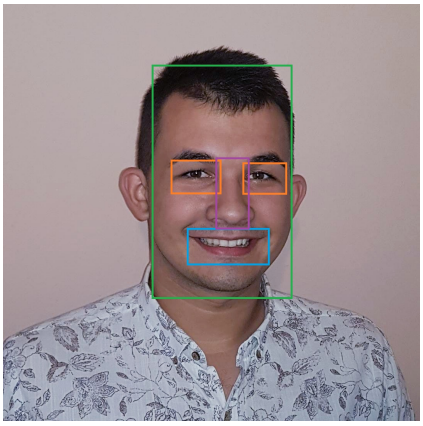
Contents

1 Introduction

This chapter aims to provide a briefly view of the project, task objectives and a short description of the structure of the paper. The program for analysing the morphological features of a human face is based on several algorithms, such as Ada Boost, and on OpenCV (Open ComputerVision).

The task of this project is to develop a program which is able to indetify the human face, and its morphological features using any programming language. In this case, the choosen programming language will be Python programming language. The objective of this program will be as follows: detect the human face in a pictures, video or a live streaming, and detecting its morphological features - eyes, nose and mouth. Face Detecion is one of the essential steps for developing Face Recognition programs, which can be see as a future development part of the program. This asspect will be discussed in the next chapters.

The project will make use of Artificial Intelligence algorithms, and will be trained on different situations where it should be able to detect if there is or is not a face and to develop its ability in the morphological features of a human face detection process. Beside each rectangle the name - Face Mouth, Eye, Nose - will be written.



In the following chapters the program functionallity will be disccused and the way the program can focus its attention on the part of the images that hold a face. Meanwhile, it is important to say that, face detection along with the detection of its morphological features is a complicate process because of the variety presented across the human race, such as position, skin color, the

presence of facial hair or glasses. Another important aspect is that we need to consider the environment problems, such as lighting conditions, and the technical details where the quality of the camera gain and image resolution is very important. In the next chapters, the way, the program can deal with the problems that may occur in the process of face detection.

2 General Aspects

2.1 Algorithms

First of all, it is important to highlight the definition of an algorithm. "An algorithm is a sequence of instructions, typically to solve a class of problems or perform a computation. Algorithms are unambiguous specification for performinf calculation, data processing, automated reasoning and other tasks."[1]

Another important aspect is that the face detection program will be based on the **Cascade Classifiers** which means that the program will use the concatenation of several classifiers using all the collected information and additional information for the next classifiers in the cascade. Trained with several hundreds "positive" sample viewsof a particular object and arbitrary "negative" images of the same size. To search for the object in the entire frame, the search windows can be moved across the classifier - process used in object or face detection and recognition. In other words, cascade classifiers are trained on a few hundreds of sample images containing the objects the program should detect, and other images that do not containg those objects.

For face detection mechanism, the fundamental algorithm that will be studied and applied in this project is **Viola-Jones** algorithm.

2.1.1 Viola-Jones Algorithm

The Viola-Jones algorithm, proposed in 2001 by Paul Viola and Michael Jones, is basically a framework for object detection that is mostly used in face detection processes. It can be trained to detec a huge variety of objects. but it is highly recomended to be used in face detection and face's morphological features applications.

This algorithm tries to solve the problem of detection the faces in an image. It is important to know that, the algorithm requires the entire face to point otwards the camera and should not be tilted to either side.

The Viola-Jones Algorithm is composed by four stages as follows:

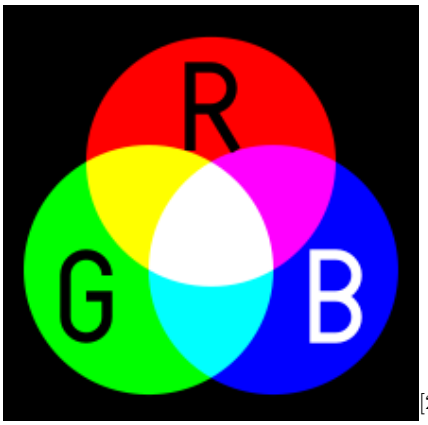
- Haar Feature Selection
- Creating an Integral Image
- AdaBoost Training

- Cascading Classifiers

The above mentionated stages will be individually discussed.

Haar Feature Selection: Haar-like features are features of a digital image that are used in object recognition algorithms. The Haar features are sequences of rescaled rectangle shaped functions. Those features come up to help reducing the cost of working with image intensities, the task of feature calculation cost. Image intesities are the RGB pixel values at each and every pixel of an image.

Talking about the RGB pixel values it is important to talk about the RGB color model. RGB color model is an additive color model, which combine three primary colors - red, green and blue - which when are added together in different quantities produces different colors. The following image represent an example of the way the RGB color model works.



[2]

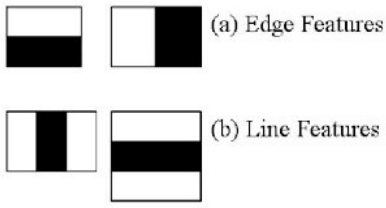
In Viola-Jones face detection framework, the algorithm works as follow: a window of target size is moved along the input image, and for each section of the image the Haar Feature is calculated, the resulted value is then compared to the already knows threshold that separates the looking-for features from the other ones. The disadvantage of using this approach it that the program needs to be trained on a big amount of test images in order to increase the accuracy of detecting the features it is supposed to find. In other words, the program increases his accuracy by learning and training on a set of test-Haar features. The advantage of a Haar-like feature over most other features is its calculation speed due to the usage of integral image. A Haar-like feature of

any size can be calculated in constant time - approximately 60 microprocessor for a 2-rectangle feature[3].

The common features of a human face, highlighted by the Haar-like features algorithm, are as follows:

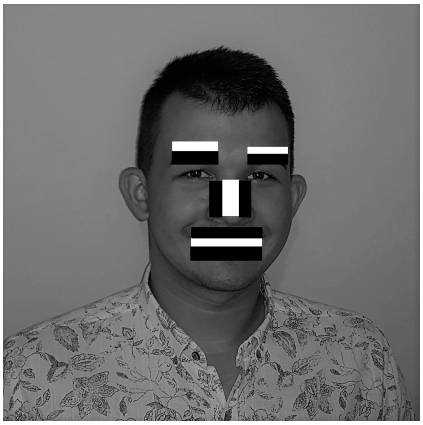
- The eyes, are represented by a darker region compared to the upper-cheeks
- The nose is a brighter region compared to the eyes or a brighter region compared to the cheeks
- There are some specific location of eyes, mouth and nose - in the way that the order is the following eyes, nose, mouth, with respect to the forehead.

The Haar-features are divided in two major classes as follows: Edge features and Line features.



[4]

An important aspect is that the ideal Haar features pixel's intensities should be 0-white and 1-black. But in the real life they are real values in the 0-1 range.Using this two features we can detect the features of a human face as in the following image - using the ideal case.



The problem that needs to be solved is the following: How can this algorithm be applied to real values? In order to have the best accuracy Violo-Jones came with a formula, whose result will be compared to the ideal case scenario. The formula contains three steps. The first one is to sum up the white pixels intensities, afterwards to sum up the black pixels intensities and to find the difference between this two using the following formula(1).

$$\Delta_{dark-white} = \frac{1}{n} * \sum_{dark}^{n} I_x - \frac{1}{n} * \sum_{white}^{n} I_x \tag{1}$$

It is important to say that, the closer the value of (1) is to 1 the chances to find a Haar-like features increase.

Creating an Integral Image: The main problem using the above mentioanted aproach is that the algorithm need to calculate the average of a certain poart several times, wich will reach an $\mathcal{O}(n^2)$ complexity. This problem can be solved by using integral images wchich will have an $\mathcal{O}(1)$ complexity.

The integral image is a concept usually found in Open CV algorithms, and it is also called Summed Area Table. It is used for quick and effective way for calculationg the sum of pixel values given in the original image, or a subsection of the image. The formula that is the base for transforming the original image to an integral image is represented by:

$$\alpha_{x,y} = Pixel_{current} + \sum Pixel_{left} + \sum Pixel_{above} \tag{2}$$

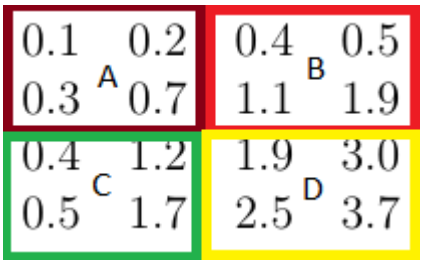
In other to have a better understanding an example is presented below. The first table represents the pixels' intensities of the original image, while the second table represents the computed value using the above mentionated formula.

0.1	0.1	0.2	0.1
0.2	0.3	0.2	0.7
0.1	0.4	0.3	0.3
0.1	0.5	0.1	0.1

Applying the the formula we have the integral image composed by the values:

0.1	0.2	0.4	0.5
0.3	0.7	1.1	1.9
0.4	1.2	1.9	3.0
0.5	1.7	2.5	3.7

The next step, after calculating the integral image is to find the value of a certain area. In this sense, the following picture will be used, which is represented by the above table which is divided in 4 areas A, B, C and D. We want to find the value of D subsection.



The formula that will be used is:

$$\Delta = A + D - B - C \tag{3}$$

By applying the (3) formula we have the following value $\Delta = 3.7 + 0.7$ - 1.9 -1.7, i.e. $\Delta = 0.8$.

Using this formula the complexity of the algorithm is $\mathcal{O}(1)$.

AdaBoost Training: Adaptive Boosting or AdaBoost is a machine learning algorithm, which is used in conjunction with many other types of learning and training algorithms in order to improve their perfomance. AdaBoost Training is a complex concept, usually combined with Decision Trees, in this chapter this concept will present its basic features and ideas, and will be widely discussed in the development part.

AdaBoost is based on three main concept. The first one is that AdaBoost is coomnoly combined with Stumps - decision trees that are composed by a root and two leaves. In other words, AdaBoost combines a lot of "weak learners" to make classification with are the stumps. In an AdaBoost mechanism, some stumps have more to say in the final classification than others. Last but not least, each stump is made by taking the previous stump's mistakes into account.

In order to understand this concept, of using forests of stump in the AdaBoost mechanism, we will use some medical data.[5]

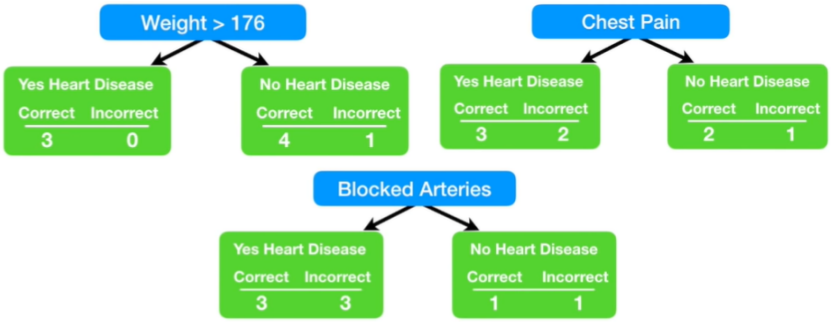
Chest Pain	Blocked Arteries	Patient Weight	Heart Disease
Yes	Yes	205	Yes
No	Yes	180	Yes
Yes	No	210	Yes
Yes	Yes	167	Yes
No	Yes	156	No
No	Yes	125	No
Yes	No	168	No
Yes	Yes	172	No

Creating a Forest of Stamps from this set of data can easily help in predicted if a patient has a heart disease or not, which is based on the patient features - chest pain, blocked arteries and patient weight. The first step is to give to each stamp a weight. For the beggining each stamp will have the same weight which is calculated by the formula $\frac{1}{number of samples}$. Hence, we get:

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	1/8
No	Yes	180	Yes	1/8
Yes	No	210	Yes	1/8
Yes	Yes	167	Yes	1/8
No	Yes	156	No	1/8
No	Yes	125	No	1/8
Yes	No	168	No	1/8
Yes	Yes	172	No	1/8

After the creation of the first stamp the sample weights will change. In the creation of the first Stamp in the forest the starting point is to check how well the Chest Pain is correlated to a Heart Disease, as follows: out of the five samples with Chest Pain, three were correctly classified as having a

Heart Disease and two were incorrectly classified, out of 3 samples without Chest Pain, two were correctly classified as not having a Heart Disease and one was incorrectly classified. Now, the same method is applied to the next two collumns - Block Arteries and Patient Weight, and the result is:



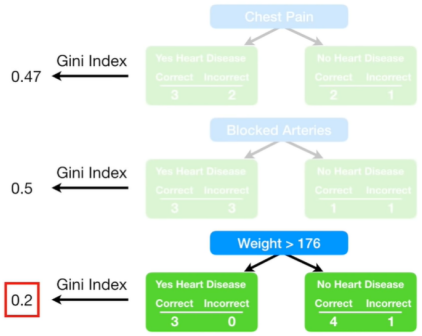
Aftwer building the Forest of Stamps, the Gini Index needs to be calculated with the formulae:

$$Gini_{left-leaf} = 1 - (probality_{correct})^2 - (probality_{incorrect})^2 \tag{4}$$

$$Gini_{right-leaf} = 1 - (probality_{correct})^2 - (probality_{incorrect})^2 \tag{5}$$

$$Gini_{total} = \frac{numbersamples_{left}}{numbersamples_{left} + numbersamples_{right}} * Gini_{left-leaf} + \frac{numbersamples_{right}}{numbersamples_{left} + numbersamples_{right}} * Gini_{right-leaf} \tag{6}$$

The next step is to find the Lowest Gini index. Hence, we have:



Which means that the Weight Stamp will be the first one in the forest. The "power" of a Stamp is simply decided by looking at the number of incorrect answears. The total error for a stump is the sum of the weights associated with the incorrectly classified samples. In this case, the total error for this stamp will be $\frac{1}{8}$. It is important to say that the sum of the stump weight should be always 1, Total error will be in the range [0,1), and 1 for a horrible stump. The "power" of a Stump is computed as follows:

$$AmountOfSay = \frac{1}{2} * \log \frac{1 - TotalError}{TotalError} \tag{7}$$

The next step that the algorithm will do is to send the forest of stumps to the Cascadind process which will be discussed next.

Cascading Classifiers: After the creation of the forest, the process will emphasize the need for the next stump to correctly classify the errors from the previous stump, by increasing it Sample Weight, and decreasing all of the other Sample Weights. The formula that is used for increasing the Sample Weight is:

$$NewSampleWeight = SampleWeight * e^{AmountOfSay} \tag{8}$$

The formula that is used for decreasing the Sample Weight is:

$$NewSampleWeight = SampleWeight * e^{-AmountOfSay} \tag{9}$$

Hence, the new table will be:

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight	New Weight
Yes	Yes	205	Yes	1/8	0.05
No	Yes	180	Yes	1/8	0.05
Yes	No	210	Yes	1/8	0.05
Yes	Yes	167	Yes	1/8	0.33
No	Yes	156	No	1/8	0.05
No	Yes	125	No	1/8	0.05
Yes	No	168	No	1/8	0.05
Yes	Yes	172	No	1/8	0.05

Now, the new Sample Weights should be normalized in order to add up to one. The normalized weight is calculated with the formula:

$$NewSampleWeight = \frac{NewSampleWeight(8,9)}{\sum NewSampleWeights(8,9)} \tag{10}$$

Hence, the final table will be:

Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Sample Weight
Yes	Yes	205	Yes	0.07
No	Yes	180	Yes	0.07
Yes	No	210	Yes	0.07
Yes	Yes	167	Yes	0.49
No	Yes	156	No	0.07
No	Yes	125	No	0.07
Yes	No	168	No	0.07
Yes	Yes	172	No	0.07

Combining this stumps the program will clearly classify the patients with a heart disease. The same process will be use for the face detection process. In other words, with integral images and boosting the algorithm's speed-increase. Cascading is used as the last step for increasing the accuracy of real-time face detection. The main idea is that most of the image region is non-face region. Instead of checking whether the window contains a face: the program will check whether it does not contain a face, because in this manner the process will be faster, by using the most relevant Haar-features.

3 Developing Tools

In this chapter, the tools that were used in developing this project will be discussed.

3.1 Python

"Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects"[6].

Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming.

3.2 OpenCV

"OpenCV (Open source computer vision) is a library of programming functions mainly aimed at real-time computer vision. Originally developed by Intel, it was later supported by Willow Garage then Itseez (which was later acquired by Intel). The library is cross-platform and free for use under the open-source BSD license. OpenCV supports the deep learning frameworks TensorFlow, Torch/PyTorch and Caffe."[7].

4 Project Development

It is important to say that for the development part it is necessay to already have installed Python, OpenCV - the second version - and PIP. To run the program, the user needs to go in the terminal or cmd, and to use the function"cd" to go in the folder the project is. after that, the Python source will be run with the command pythonSource.py - supposing the source file is name pythonSource.py.

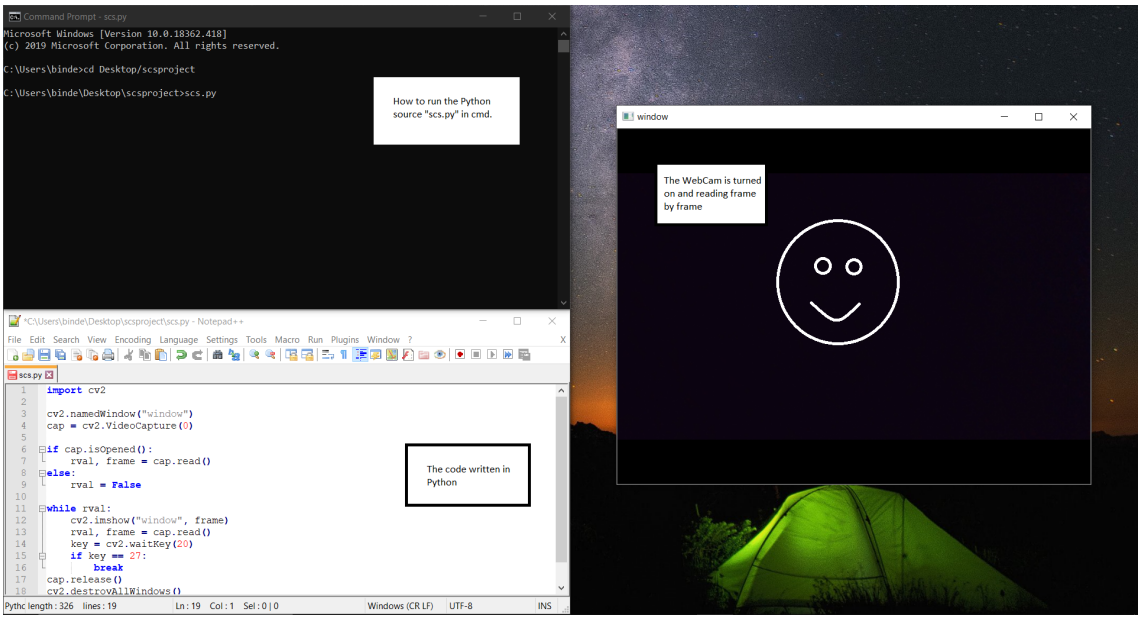
4.1 Hardware Connection

The first step in the development, is to make the connection between the program and the hardware, which, in this project is represented by the WebCam. In this sense, the cv2 library was imported and the following functions where used:

- cv2.namedWindow("window"): this function creates a window called "window" and load the image to it.
- cv2.VideoCaputre(0): this function capture the image receive from the WebCame frame by frame. The argument can be an index or a name of a video-file. The argument 0 means that the program will check for the first camera. Remark: In order to avoid the re-openings of the WebCam, it is allocated to the object "cap" which will open it just one time at the beginning of the program.
- cv2.VideoCapture(0).isOpened(): this function checks if the WebCam was open correctly or not.
- cv2.VideoCapture(0).read(): this function capture the image frame by frame.
- cv2.imshow("window", frame): this function display the resulted frame.
- cv2.waitKey(20): this function shows the speed of the video.
- cv2.VideoCapture(0).release(): this function release the capture.
- cv2.destroyAllWindows(): this function destroys all open windows.

- if key == 27: this function activate when the 27 ASCII code key is pressed, i.e. the ESC key is pressed

The following image represents the way the program is run and what it will display.



It is important to say that, the user will be able to select an image saved on the Computer System, without turning on the camera. In order to exit the program the user needs to press "ESC" key, in this way the program will be turned off.

4.2 Face Detection

In this part, the way the face is detected will be presented. As it was mentionated in the above chapter, the ideal case is when the image is represented by black and white pixels, but in the real life this case is not possible. In order to get closer to the idel case we will use the cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY) function, which converts the real image to an gray scale image. The code is presented in the next figure.

```
while rval:
    cv2.imshow("window", cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY))
    rval, frame = cap.read()
    key = cv2.waitKey(20)
    if key == 27:
        break
```

It is important to know how to convert a real colored frame into an RGB scaled frame. In this sense, the following image represents the code that transform the frame into a RGB scaled frame.

```
11  while rval:
12      cv2.imshow("window", cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
13      rval, frame = cap.read()
14      key = cv2.waitKey(20)
15      if key == 27:
16          break
```

An example is presented below. The picture on the left, represents the original frame, received by the WebCam while, the picture in the middle represents the frame filtered through the gray-scale filter, and the picture on the right represents the same frame, but after it was filtered through the RGB-scale filter.

In order to use Haar Cascade Clasifiers, OpenCV comes with a lot of pretrained classifiers - for eyes, face, nose, etc. These clasifiers comes under the form of .xml files and there are found in opencv/data/haarcascade/[8] folder. For example, loading the default frontal face classifier, the following piece of code is used:

```
haar_cascade_face = cv2.CascadeClassifier('data\haarcascade_frontalface_default.xml')
```

With all of this being said, the developing process will start with the Face detection mechanism which will be discussed next. The detectMultiscale moduleof the classifiers will be used. The module will return a rectangle with coordinates (x, y, w, h) that will be placed around the face. It is important to say that the module has two important arguments. The first one is the scalefactor and the second one the minNeighbors. The first one, refer to the fact that, in a frame there could be more images, ones that could be closer to the camera, and others that can be in the back of the image. This factor is the compensator. The minNeighbors specifies the number of neighbors a rectangle should have in order to have all the properties of a face.The following picture represents an example of the way the above function works.

```
faces_rects = haar_cascade_face.detectMultiScale(frame, scaleFactor = 1.5, minNeighbors = 4);
```

The code for face detection is represented in the following image, code which will be explained in a little bit.

```
while True:
    key=cv2.waitKey(20)
    cv2.imshow('window',img)
    face_coordinates = haar_cascade_face.detectMultiScale(gray, scaleFactor = 1.2, minNeighbors = 5);
    eye_coordinates= haar_cascade_eye.detectMultiScale(gray, scaleFactor = 1.5, minNeighbors = 4);
    nose_coordinates= haar_cascade_nose.detectMultiScale(gray, scaleFactor = 1.3, minNeighbors = 5);
    mouth_coordinates= haar_cascade_mouth.detectMultiScale(gray, scaleFactor = 2.1, minNeighbors = 5);
    for (x1,y1,w1,h1) in face_coordinates:
     cv2.rectangle(img, (x1, y1), (x1+w1, y1+h1), (0, 255, 0), 2)
     cv2.putText(img,'Face',(x1+w1,y1+h1), font, 2, (0, 255, 0), 2, cv2.LINE_AA)
    for (x1,y1,w1,h1) in mouth_coordinates:
     cv2.rectangle(img, (x1, y1), (x1+w1, y1+h1), (255, 255, 0), 2)
     cv2.putText(img,'Mouth',(x1+w1,y1+h1), font, 2, (255, 255, 0), 2, cv2.LINE_AA)
    for (x2,y2,w2,h2) in eye_coordinates:
     cv2.rectangle(img, (x2, y2), (x2+w2, y2+h2), (255, 0, 0), 2)
     cv2.putText(img,'Eye',(x2+w2,y2+h2), font, 2, (255, 0, 0), 2, cv2.LINE_AA)
    for (x3,y3,w3,h3) in nose_coordinates:
     cv2.rectangle(img, (x3, y3), (x3+w3, y3+h3), (0, 0, 255), 2)
     cv2.putText(img,'Nose',(x3+w3,y3+h3), font, 2, (0, 0, 255), 2, cv2.LINE_AA)
    if key == 27:
        break
cv2.destroyAllWindows()
```

The first line of code, represents the part of the code that will import the OpenCV library - cv2- where most of the functions thaat were used in this project can be found. The next step is to define the font of the text that will be used for showing the detected parts of the face, in this case the font that will be use is HERSHEY_SIMPLEX. The Haar Cascade classifier that will be used in the developing of this project is found in the folder "data haarcascade_frontalface_default.xml". The line 5 will name the Window as the function says "window". The following libe of code, represent that the object cap, will refer to the caption of the first WebCam. The next part of code, between line 8-11 checks if the WebCam is open in the right way and will assign the value read from the WebCam to the "frame" object, giving to rval the value of true. If the WebCam is not open, rval will be false and the program's execution will stop. Between the lines 13-20, in the window "window" the frames read from the WebCam are displayed and new frames are read at the same time. Face_coordinates will be an array of coordinates of type (x, y, w, h) storing the coordinates that shows the face in each frame. The for loop will check for the tuples (x, y, w, h) in the face_coordinates arrays and draw the rectangle around the face, a green rectangle of thickness 2, adding the ford "Face" near the rectangle. If the ESK key is pressed the while loop will break and the function will release and destroy all the open windows.

4.3 Eyes Detection

The next step is to detenct the eyes. The process will be detailed explained next.

```
while True:
    key=cv2.waitKey(20)
    cv2.imshow('window',img)
    face_coordinates = haar_cascade_face.detectMultiScale(gray, scaleFactor = 1.2, minNeighbors = 5);
    eye_coordinates= haar_cascade_eye.detectMultiScale(gray, scaleFactor = 1.5, minNeighbors = 4);
    nose_coordinates= haar_cascade_nose.detectMultiScale(gray, scaleFactor = 1.3, minNeighbors = 5);
    mouth_coordinates= haar_cascade_mouth.detectMultiScale(gray, scaleFactor = 2.1, minNeighbors = 5);
    for (x1,y1,w1,h1) in face_coordinates:
     cv2.rectangle(img, (x1, y1), (x1+w1, y1+h1), (0, 255, 0), 2)
     cv2.putText(img,'Face',(x1+w1,y1+h1), font, 2, (0, 255, 0), 2, cv2.LINE_AA)
    for (x1,y1,w1,h1) in mouth_coordinates:
     cv2.rectangle(img, (x1, y1), (x1+w1, y1+h1), (255, 255, 0), 2)
     cv2.putText(img,'Mouth',(x1+w1,y1+h1), font, 2, (255, 255, 0), 2, cv2.LINE_AA)
    for (x2,y2,w2,h2) in eye_coordinates:
     cv2.rectangle(img, (x2, y2), (x2+w2, y2+h2), (255, 0, 0), 2)
     cv2.putText(img,'Eye',(x2+w2,y2+h2), font, 2, (255, 0, 0), 2, cv2.LINE_AA)
    for (x3,y3,w3,h3) in nose_coordinates:
     cv2.rectangle(img, (x3, y3), (x3+w3, y3+h3), (0, 0, 255), 2)
     cv2.putText(img,'Nose',(x3+w3,y3+h3), font, 2, (0, 0, 255), 2, cv2.LINE_AA)
    if key == 27:
        break
cv2.destroyAllWindows()
```

The first highlighted line of code is the line that give to the haar_cascade_eye the ".xml" file that contains the Haar-Cascade classifiers for finding the eyes. The same method is applied. We have an array of coordinates in eye_coordinates and the rectangle will be around the eyes writing "Eye" beside them.

4.4 Mouth Detection

The next step is to detenct the mouth. The process will be detailed explained next.

```
while True:
    key=cv2.waitKey(20)
    cv2.imshow('window',img)
    face_coordinates = haar_cascade_face.detectMultiScale(gray, scaleFactor = 1.2, minNeighbors = 5);
    eye_coordinates= haar_cascade_eye.detectMultiScale(gray, scaleFactor = 1.5, minNeighbors = 4);
    nose_coordinates= haar_cascade_nose.detectMultiScale(gray, scaleFactor = 1.3, minNeighbors = 5);
    mouth_coordinates= haar_cascade_mouth.detectMultiScale(gray, scaleFactor = 2.1, minNeighbors = 5);
    for (x1,y1,w1,h1) in face_coordinates:
     cv2.rectangle(img, (x1, y1), (x1+w1, y1+h1), (0, 255, 0), 2)
     cv2.putText(img,'Face',(x1+w1,y1+h1), font, 2, (0, 255, 0), 2, cv2.LINE_AA)
    for (x1,y1,w1,h1) in mouth_coordinates:
     cv2.rectangle(img, (x1, y1), (x1+w1, y1+h1), (255, 255, 0), 2)
     cv2.putText(img,'Mouth',(x1+w1,y1+h1), font, 2, (255, 255, 0), 2, cv2.LINE_AA)
    for (x2,y2,w2,h2) in eye_coordinates:
     cv2.rectangle(img, (x2, y2), (x2+w2, y2+h2), (255, 0, 0), 2)
     cv2.putText(img,'Eye',(x2+w2,y2+h2), font, 2, (255, 0, 0), 2, cv2.LINE_AA)
    for (x3,y3,w3,h3) in nose_coordinates:
     cv2.rectangle(img, (x3, y3), (x3+w3, y3+h3), (0, 0, 255), 2)
     cv2.putText(img,'Nose',(x3+w3,y3+h3), font, 2, (0, 0, 255), 2, cv2.LINE_AA)
    if key == 27:
        break
cv2.destroyAllWindows()
```

The first highlighted line of code is the line that give to the haar_cascade_mouth the ".xml" file that contains the Haar-Cascade classifiers for finding the eyes. The same method is applied. We have an array of coordinates in eye_coordinates and the rectangle will be around the mouth writing "Mouth" beside them.

4.5 Nose Detection

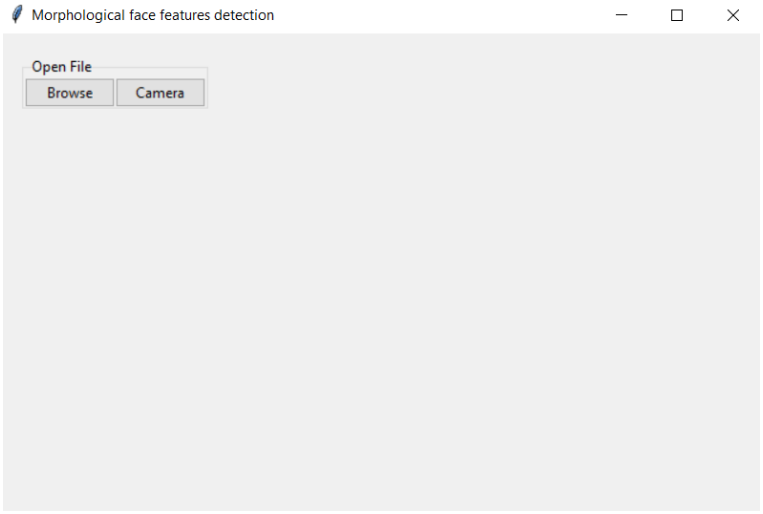
The next step is to detenct the nose. The process will be detailed explained next.

```
while True:
    key=cv2.waitKey(20)
    cv2.imshow('window',img)
    face_coordinates = haar_cascade_face.detectMultiScale(gray, scaleFactor = 1.2, minNeighbors = 5);
    eye_coordinates= haar_cascade_eye.detectMultiScale(gray, scaleFactor = 1.5, minNeighbors = 4);
    nose_coordinates= haar_cascade_nose.detectMultiScale(gray, scaleFactor = 1.3, minNeighbors = 5);
    mouth_coordinates= haar_cascade_mouth.detectMultiScale(gray, scaleFactor = 2.1, minNeighbors = 5);
    for (x1,y1,w1,h1) in face_coordinates:
     cv2.rectangle(img, (x1, y1), (x1+w1, y1+h1), (0, 255, 0), 2)
     cv2.putText(img,'Face',(x1+w1,y1+h1), font, 2, (0, 255, 0), 2, cv2.LINE_AA)
    for (x1,y1,w1,h1) in mouth_coordinates:
     cv2.rectangle(img, (x1, y1), (x1+w1, y1+h1), (255, 255, 0), 2)
     cv2.putText(img,'Mouth',(x1+w1,y1+h1), font, 2, (255, 255, 0), 2, cv2.LINE_AA)
    for (x2,y2,w2,h2) in eye_coordinates:
     cv2.rectangle(img, (x2, y2), (x2+w2, y2+h2), (255, 0, 0), 2)
     cv2.putText(img,'Eye',(x2+w2,y2+h2), font, 2, (255, 0, 0), 2, cv2.LINE_AA)
    for (x3,y3,w3,h3) in nose_coordinates:
     cv2.rectangle(img, (x3, y3), (x3+w3, y3+h3), (0, 0, 255), 2)
     cv2.putText(img,'Nose',(x3+w3,y3+h3), font, 2, (0, 0, 255), 2, cv2.LINE_AA)
    if key == 27:
        break
cv2.destroyAllWindows()
```

The first highlighted line of code is the line that give to the haar_cascade_eye the ".xml" file that contains the Haar-Cascade classifiers for finding the nose. The same methodas before is applied. We have an array of coordinates in nose_coordinates and the rectangle will be around the nose writing "Nose" beside them.

4.6 Browsing through the computer

In this chapter, an additional feature for the application will be presented. The user can choose from using the camera for real time recognition or he can upload a picture from the computer memory. In this sense, the following user interface will be used.



It is important to remark that, more about the user interface will be explained in the following chapter.

4.7 Final Code

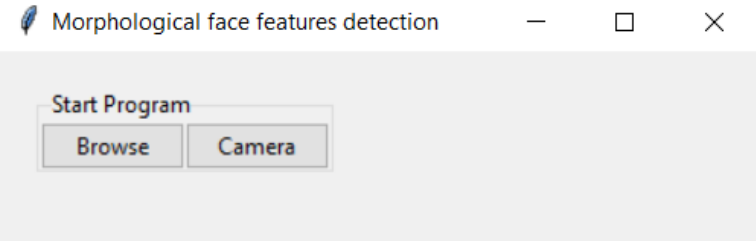
With everything said and explained, the final version of code which will implement the "Analysis program of the face's morphological features", using the Computer's Camera, will be showed in the next figure.

```
import cv2
font = cv2.FONT_HERSHEY_SIMPLEX
haar_cascade_face = cv2.CascadeClassifier('data\haarcascade_frontalface_default.xml')
haar_cascade_eye = cv2.CascadeClassifier('data\haarcascade_eye.xml')
haar_cascade_mouth = cv2.CascadeClassifier('data\haarcascade_mouth.xml')
haar_cascade_nose = cv2.CascadeClassifier('data\haarcascade_nose.xml')
cv2.namedWindow("window")
cap = cv2.VideoCapture(0)
if cap.isOpened():
    rval, frame = cap.read()
else:
    rval = False
while rval:
    cv2.imshow("window", frame)
    rval, frame = cap.read()
    key = cv2.waitKey(20)
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    face_coordinates = haar_cascade_face.detectMultiScale(gray, scaleFactor=1.2, minNeighbors=5);
    eye_coordinates = haar_cascade_eye.detectMultiScale(gray, scaleFactor=1.5, minNeighbors=4);
    nose_coordinates = haar_cascade_nose.detectMultiScale(gray, scaleFactor=1.3, minNeighbors=5);
    mouth_coordinates = haar_cascade_mouth.detectMultiScale(gray, scaleFactor=1.7, minNeighbors=11);
    for (x1,y1,w1,h1) in face_coordinates:
     cv2.rectangle(frame, (x1, y1), (x1+w1, y1+h1), (255, 0, 0), 2)
     cv2.putText(frame,'Face',(x1+w1,y1+h1), font, 2, (255, 0, 0), 2, cv2.LINE_AA)
    for (x2,y2,w2,h2) in eye_coordinates:
     cv2.rectangle(frame, (x2, y2), (x2+w2, y2+h2), (255, 255, 0), 2)
     cv2.putText(frame,'Eye',(x2+w2,y2+h2), font, 2, (255, 255, 0), 2, cv2.LINE_AA)
    for (x3,y3,w3,h3) in nose_coordinates:
     cv2.rectangle(frame, (x3, y3), (x3+w3, y3+h3), (255, 0, 255), 2)
     cv2.putText(frame,'Nose',(x3+w3,y3+h3), font, 2, (255, 0, 255), 2, cv2.LINE_AA)
    for (x4,y4,w4,h4) in mouth_coordinates:
     y4 = int(y4 - 0.15 * h4)
     cv2.rectangle(frame, (x4, y4), (x4+w4, y4+h4), (0, 255, 0), 2)
     cv2.putText(frame,'Mouth',(x4+w4,y4+h4), font, 2, (0, 255, 0), 2, cv2.LINE_AA)
    if key == 27:
        break
cap.release()
cv2.destroyAllWindows()
```

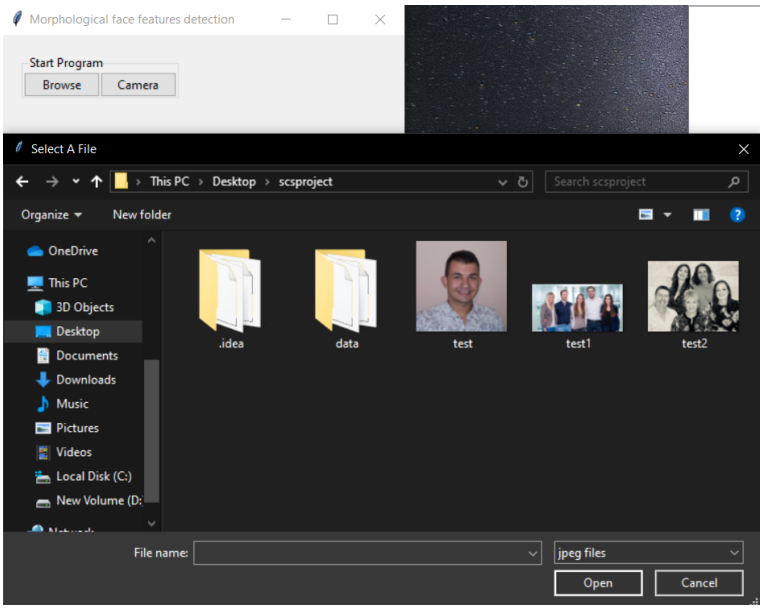
At the end of this chapter, it is important to mention that the program could have different upgrades. The main use of this code could be in the developing of a face recognition program, program which could keep track of the "wave" of persons that enter a building or to a concert. In other words, there are several ways of improving this program.

5 User Interface

The user interface, UI of this project helps the user to interract with the program. In this case, there are two options: to turn on the camera, and to choose a picture from the computer's memory. The next picture represents how the user interface looks like.



When pressing the Browse file, a window will appear giving the user the possibility to choose a file from the memory.



After the user open the file "behind the scenes" the OS will execute a cmd command with the form:

```
command = "C:\\Users\\binde\\Desktop\\scsproject\\test.py "
command=command+self.filename
os.system(command)
```

If the user press the Camera Button the command will be as follows:

```
command = "C:\\Users\\binde\\Desktop\\scsproject\\scs.py "
os.system(command)
```

The final code for the User Interface is the following one:

```
from tkinter import *
from tkinter import ttk
from tkinter import filedialog
import inspect
import os

class Root(Tk):
    def __init__(self):
        super(Root, self).__init__()
        self.title("Morphological face features detection")
        self.minsize(400, 100)

        self.labelFrame = ttk.LabelFrame(self, text="Start Program")
        self.labelFrame.grid(column=0, row=1, padx=20, pady=20)

        self.button()

    def button(self):
        self.button = ttk.Button(self.labelFrame, text="Browse", command=self.fileDialog)
        self.button.grid(column=1, row=1)
        self.button = ttk.Button(self.labelFrame, text="Camera", command=self.camera)
        self.button.grid(column=3, row=1)

    def fileDialog(self):
        self.filename = filedialog.askopenfilename(initialdir="/", title="Select A File", filetype=
        (("jpeg files", "*.jpg"),("jpeg files", "*.jpeg"), ("all files", "*.*")))
        self.label = ttk.Label(self.labelFrame, text="")
        self.label.grid(column=1, row=2)
        command = "C:\\Users\\binde\\Desktop\\scsproject\\test.py "
        command=command+self.filename
        os.system(command)

    def camera(self):
        command = "C:\\Users\\binde\\Desktop\\scsproject\\scs.py "
        os.system(command)


root = Root()
root.mainloop()
```

The first five lines of code represent the libraries that will be used. First three lines are libraries from Tkinter, which was used for developing the interface. The fifth one is the library that helps the program to open the cmd and execute the command. The program will start by executing the Root(Tk) as the main loop. The initialization method will construct the

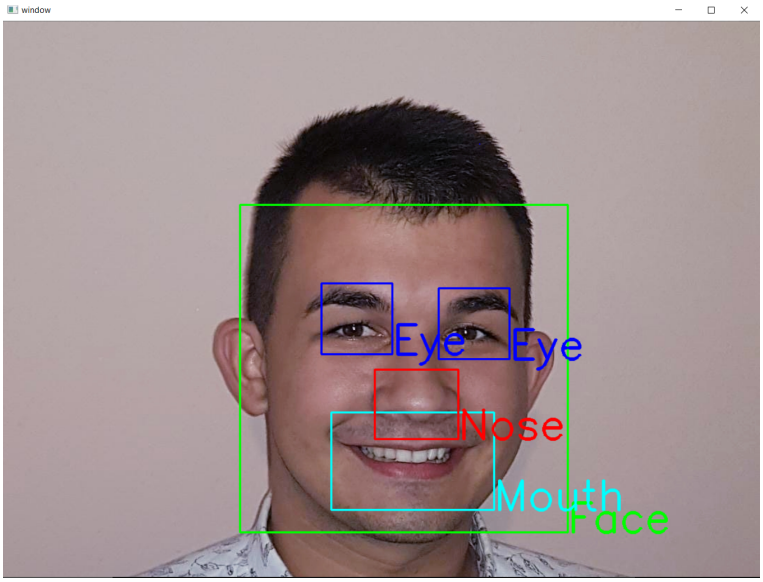
initial frame. The button method, initializes the buttons that the user will interract with, each one of them doing its own job. The fileDialog method is the method that will help the user browse in the computer memory, mehtod that is used by the Browse buton. The method camera fill be used by the Camera button, turning on the camera for the program. The command is a String that represents the command that will be sent by the interface tothe command prompt. The function os.system(command) will execute the given command.

6 Test cases

In this chapter, the program will be tested on some cases, such as how the program reacts on different pictures, with one person or more persons. The code for the testing cases is the following:

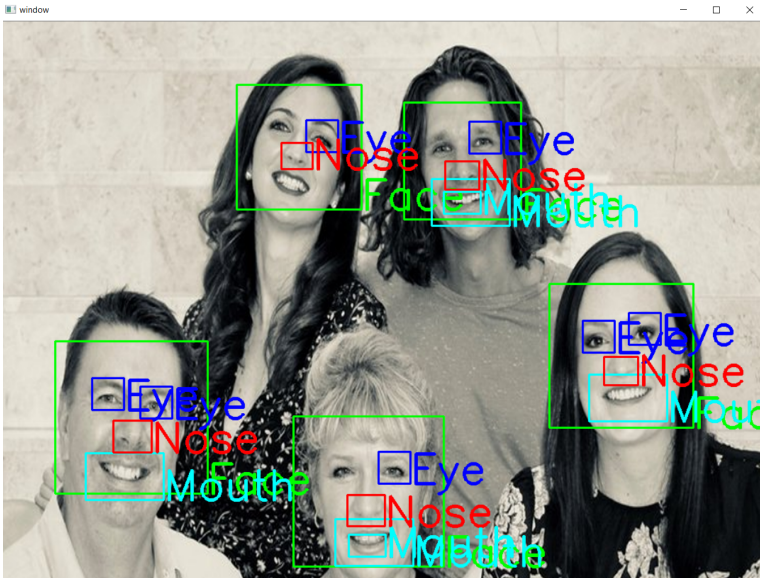
```
import cv2
import time
import sys

font = cv2.FONT_HERSHEY_SIMPLEX
img = cv2.resize(cv2.imread(sys.argv[1]),(1080,1080),interpolation = cv2.INTER_CUBIC)
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
haar_cascade_face = cv2.CascadeClassifier('data\haarcascade_frontalface_default.xml')
haar_cascade_eye = cv2.CascadeClassifier('data\haarcascade_eye.xml')
haar_cascade_mouth = cv2.CascadeClassifier('data\haarcascade_mouth.xml')
haar_cascade_nose = cv2.CascadeClassifier('data\haarcascade_nose.xml')

while True:
    key=cv2.waitKey(20)
    cv2.imshow('window',img)
    face_coordinates = haar_cascade_face.detectMultiScale(gray, scaleFactor = 1.2, minNeighbors = 5);
    eye_coordinates= haar_cascade_eye.detectMultiScale(gray, scaleFactor = 1.5, minNeighbors = 4);
    nose_coordinates= haar_cascade_nose.detectMultiScale(gray, scaleFactor = 1.3, minNeighbors = 5);
    mouth_coordinates= haar_cascade_mouth.detectMultiScale(gray, scaleFactor = 2.1, minNeighbors = 5);
    for (x1,y1,w1,h1) in face_coordinates:
     cv2.rectangle(img, (x1, y1), (x1+w1, y1+h1), (0, 255, 0), 2)
     cv2.putText(img,'Face',(x1+w1,y1+h1), font, 2, (0, 255, 0), 2, cv2.LINE_AA)
    for (x1,y1,w1,h1) in mouth_coordinates:
     cv2.rectangle(img, (x1, y1), (x1+w1, y1+h1), (255, 255, 0), 2)
     cv2.putText(img,'Mouth',(x1+w1,y1+h1), font, 2, (255, 255, 0), 2, cv2.LINE_AA)
    for (x2,y2,w2,h2) in eye_coordinates:
     cv2.rectangle(img, (x2, y2), (x2+w2, y2+h2), (255, 0, 0), 2)
     cv2.putText(img,'Eye',(x2+w2,y2+h2), font, 2, (255, 0, 0), 2, cv2.LINE_AA)
    for (x3,y3,w3,h3) in nose_coordinates:
     cv2.rectangle(img, (x3, y3), (x3+w3, y3+h3), (0, 0, 255), 2)
     cv2.putText(img,'Nose',(x3+w3,y3+h3), font, 2, (0, 0, 255), 2, cv2.LINE_AA)
    if key == 27:
        break
cv2.destroyAllWindows()
```

It is important to say that the program may have some failures, depending on the picture quality, lightning, face position and face size. For example, if you have a good quality, favorable size face picture we will receive a favorable result, as follows.



There could be moments when the program may fail in delivering the right result because of the resolution of the given picture. Such an example is the following picture.[9]

One of the main problem here is the image quality the size, and the face position. For example, in the next figure, the program return less errors, regarding to the last on.



In conclusion, regarding to the testing chapter, we can say that, the program may have some errors due to the picture resolution, lighting and other picture features.

7 Conclusion

In conclusion, this paper explain the mechanism of developing "from scratch" of an "Analysis program of the face's morphological features" by using Python as programming language, and OpenCV libraries, and the Haar-Cascade classifiers defined in the OpenCV libraries. This program detects only the basic features of a face, such as nose, mouth eyes and the face itself, but it could be develop to solve more complex projects. It is important to say that, due to the fact that the problem receive real time images, there is a possibility for the program accuracy to do not be as high as in a given picture. In other words we may have some errors in the detection, such as mouth detection.

8 Bibliography

[1] https://en.wikipedia.org/wiki/Algorithm

[2] https://en.wikipedia.org/wiki/RGB_color_model#/media/File:AdditiveColor.svg

[3]https://en.wikipedia.org/wiki/Haar-like_feature

[4]https://stackoverflow.com/questions/23171232/viola-jones-algorithm-sum-of-pixels

[5]https://www.youtube.com/watch?v=LsK-xG1cLYA

[6]https://en.wikipedia.org/wiki/Python_(programming_language)

[7]https://en.wikipedia.org/wiki/OpenCV

[8]https://github.com/opencv/opencv/tree/master/data/haarcascades

[9]https://www.google.com

[10]https://www.google.com/url?sa=i&source=images&cd=&ved=2ahUKEwj86NWJsa3mAhXwKHRUTDJMQjRx6BAgBEAQ&url=https%3A%2F%2Fmelbourneshouldergroup.com%2F&psig=jlAugAzsCRubf&ust=1576146408736740