École Polytechnique Montréal
Génie Informatique

INF6804 : Vision par ordinateur
Présenté à Guillaume-Alexandre Bilodeau

Laboratoire 2

Travail remis par :
XING, Jin Ling- #1915481
MICHÉA, Luc - #1921803

1 Mars 2019

1 Introduction

In this lab, we are going to use two different techniques of classification to do a face tracking experiment. We want to see how efficient one technique is compared to another on different conditions. So first, we will explain the two different methods, how they work, what they do. Then, we will set down some requirements in which the two methods should work and assume which one will work better in those conditions. Finally, we will test our methods in those condition, evaluate the results and analyse them.

2 Identification and explanation

First of all we want to identify something in a picture and then identify if it's the same object that we had in the previous picture. To do that we could use several methods, one of the most powerful one would be the SIFT method because of her incredible results in this specific case of work. Unfortunately, SIFT has a patent and can't be used freely. Just for information, SIFT is based on different point extracted on a picture and their neighbour. The power of this method resides in the fact that you only need a few points to detect an object. In this lab, we will use two other methods to detect and track the faces of people in a video. The first method is based on the Histogram of Oriented Gradients (**HOG**) and so the different edges of an object in a picture whereas the second method is based on Local Binary Patterns (**LBP**) and so on the different textures of objects in a picture. We will now explain those two methods :

2.1 Histogram of Oriented Gradients

During this Lab, we are going to use the HOG (Histogram of Oriented Gradients) algorithm. It is based on the gradient of a picture. Basically, the Gradient of a picture is a Vector going from White Pixels to Black Pixels.

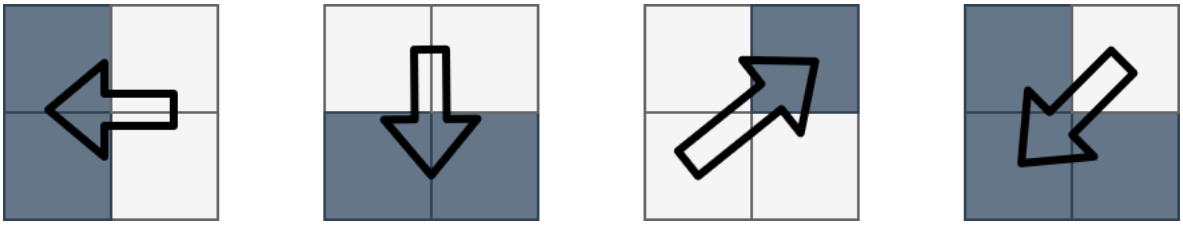

Fig. 1 – Illustration of a gradient of a picture

Then, from those Gradients we extract the different angles and we create an histogram containing the angles describing the object we want to detect. Finally, we develop a neural network that works to understand if a new histogram describes the object we want to detect or not. (Most of the time, the approximation on the angles are of around 8 degrees.) So for example, we will use a special HoG contained in a library in this lab. We will use **dlib**'s face recognition HoG trained neural network to find faces in pictures. This HoG Method is trained on around 3 000 pictures (Half positives, half negatives) on different positions of heads (front, profile, side and different mirroring to have the complementary).

2.2 Local Binary Pattern

An older an more simple method could be to search for similar patterns of two objects. It's a really simple and fast method because we can use binary numbers to describe a pattern in a picture. If we take a simple example, for a 3x3 bloc of pixels we can determine the pattern as follows :

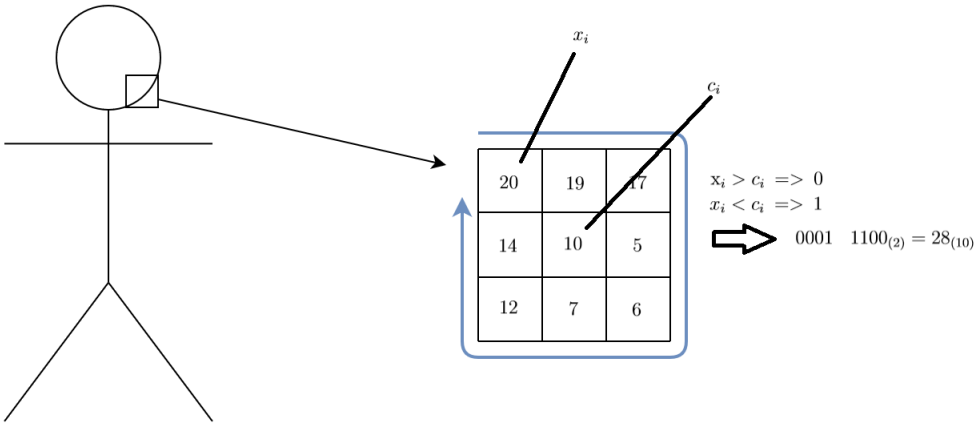

Fig. 2 – Example of the determination of the pattern

Then from this binary number we transform it into a decimal number. And we create an histogram with all those numbers. Now, from the histogram of our object, we can use the histogram of a new picture. If in the new histogram we find the same pattern it means that our object is indeed in the picture, else it means our object is not in the picture. Finding the same pattern can be defined using the Hamming distance definition with the histograms. Unlike HOG we don't have to train any Neural Networks. Example of the histograms :

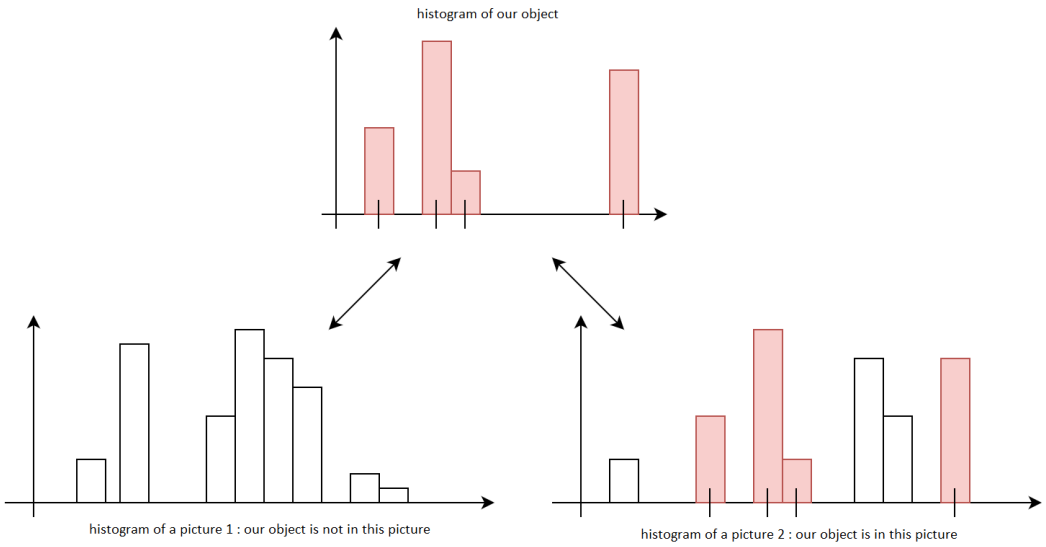

Fig. 3 – Example of the detection of an object using histograms

Once we found the same pattern, we can deduce that every pixels of the histogram of the picture that belongs to our pattern is a pixel of our object. The LBP we will use during this lab is a face recognition application implemented in OpenCV. Face recognition was a really common application for LBP just because the face has some special features such as eyes, nose... that differentiate it from a lot of objects.

2.3 Specific cases

So, during this lab we want to create certain situations that will challenge the two algorithms we will try to guess which one is best in which case and then we will do some experiments to either prove our hypothesis or make a discussion why it didn't work.

- Illumination Variation :
  In this case, the illumination doesn't change when the time passing, but the face of the man moved between

sun and sunshade. The aim of this video is to be complex enough to make the LBP algorithm to think there's the man's face (what we want to detect). The reason is because LBP needs the neighbour's pixels to create evaluation histogram, but neighbour's pixels changed. We can conclude that in this case, the HOG will be better at detecting the human face in the picture. The idea of this experiment went from the fact that if we want to detect the human face between sun and sunshade ; then the HOG algorithm will not really be affected by this brutal change, because if will only affect part of the vectors and their weight, but most of the vectors will still discriminate the face from the background. The LBP will perform better when the man is fully in the sun or in the sunshade.

- Occlusion :
  In this case we want to see if the algorithms can adapt themselves to a situation where the man's face is occluded by his hand during the period of the tracking. In this case we can guess that the best algorithm would be the LBP method, although maybe both LBP and HOG will not perform well. For the HOG, the HOG can't adapt itself to occlusion, because the gradient will be totally different and it's not based on the pixels themselves but more on the shape of things. So, the HOG may not be able to detect a distorted shape with hands on the man's face, but will be able to detect the shape when the hand is moving away. For LBP algorithm, the textures between the hand and face are pretty much similar but the small details still lost a lot, such as the human nose, human eyes and human face shape.

- Scale Changes :
  In this case the human face is without occlusion but the human will have different scales. We really think that HOG will perform better in tracking. The gradients of human face will not change so much, because the shape of human face doesn't change. We although expect that the LBP method would not be as good as HoG method, because pixels inside the human face will change (some pixels will most likely disappear when others may appear), then when the LBP compared the neighbour's pixels, it will not be the same evaluation histogram.

3 Experiments

3.1 Data-set used

All the data-sets we used are in the website "VOT challenge", we chose three of them to do the tests in different cases.

- Illumination Variation :
  The perfect example to show this has been seen in "VOT challenge". It is the *"Sunshade"* testing file. There is a man who is moving between sun and sunshade, the illumination didn't change during the video time.

- Occlusion :
  In the *"Occlusion"* file we can find a *"Hand(VOT2018)"* file that seems to be perfect to measure the track of human face with each methods when the man's hand is moving around his face.

- Scale Changes :
  In the data-sets, we can see some people moving videos, one of those seems really interesting and is the *"Book(VOT2018)"* video. It is really interesting because the woman is moving back and forth in the picture plan, and her hands have a book, but the book doesn't affect us to track her face.

3.2 Evaluation criteria

In this lab our evaluation criteria is a bit different from what you could expect. In fact, our aim here is to detect an object and track it. So, we don't really think that the ground-truths given in the Data-sets files will be relevant for this problem. So, we decided to create our own criteria based on several things. First of all, we need to present every variables :

- **N** : It's simply the number of pictures in the DataSet.

- **tracking_error** : It's the number of pictures where ; either we didn't detect any face in the picture (because in every of our datasets, the faces of the subjects are always inside the picture) or the number of pictures where the rectangle was supposed not to be the subject. We will see later what it means.
- **tracking_ok** : It represents everytime we got a rectangle and it represented the subject.
- **tracking_complete** : It's the percentage of pictures where the subject was completely tracked.
- **tracking_uncomplete** : It's the percentage of picures where the subject wasn't tracked.
- **rects_list** : It's a list containing every rectangles detected from the method.
- **rect_obj_list** : It's a list containing every rectangles detected from the method that also where considered as our subject and tracked.
- **tracking_obj_detected** : It's the percentage of rectangles detected that where considered as our subject in the end.

Then our evaluation criteria can be calculated as follows :

```
####
##  evaluation criteria
####

tracking_complete=float(tracking_ok)/N
tracking_uncomplete=float(tracking_error)/N
tracking_obj_detected=len(rect_obj_list)/len(rects_list)
```

Now, how did we defined if the rectangle really tracked our subject or not. Well, we used a simple reasoning which can be illustrated. We considered that if the method detected the subject on every pictures, then the distance of the middle of the detection rectangle will not be too far between two pictures. Also, we defined the fact that our object will not change it's size so much between two pictures, so that if we detect something else in the picture that is much bigger, then we will not take that into account. We can illustrate that as follows :

Fig. 4 – Example of tracking

Here, the method detected two different rectangles on the second picture. The red rectangle is our subject, we see that the distance between the two rectangles is not really high, and that the size didn't change much. On the opposite, the yellow rectangle represents what we shouldn't start tracking by mistake.

4 implementation of the methods

First of all, we can say that the steps of the two algorithms are really similar for some parts, that is why we will explain most of the parts for HOG and then assume that if it's not precised, the parts are the same. Then, most of the imports will not be shown, if you want to see them, you can find them in the .zip file.

4.1 Method 1 : Histogram of Oriented Gradients

4.1.1 Implementation

First of all, for both cases, we import the pictures, and we put them in a list. Because the HOG method implemented in the **dlib** only takes RGB or 8bits grayscale and **OpenCV** image reading is in BGR format we have to change the image to a grayscale. Then, we set some basic parameters for the rest of the algorithm.

```
####
## Pictures import
####

#import images
#filenames = [img for img in glob.glob("C:/Users/lucmi/OneDrive/Documents/INF6804/Datasets/VOT2013/Illumination/*.jpg")]
#filenames = [img for img in glob.glob("C:/Users/lucmi/OneDrive/Documents/INF6804/Datasets/VOT2013/Obstruction/*.jpg")]
filenames = [img for img in glob.glob("C:/Users/lucmi/OneDrive/Documents/INF6804/Datasets/VOT2013/Size/*.jpg")]
filenames.sort() # ADD THIS LINE


imlist = []
for img in filenames:
    n= cv2.imread(img)
    n= cv2.cvtColor(n, cv2.COLOR_BGR2GRAY)
#    n= dlib.load_rgb_image(img)
    imlist.append(n)

#parameters of images
im = imlist[0] #first image
height = np.size(im, 0)
width = np.size(im, 1)
#RGB = np.size(im, 2)
N = len(imlist)
```

Then we set the detector and use it, in this step for optimisation matter we also decide to do the tracking part of the algorithm.

```
####
## Initialization of the detector
####

hogFaceDetector = dlib.get_frontal_face_detector()
#faceRects = hogFaceDetector(frameDlibHogSmall, 0)
#hog = cv2.HOGDescriptor()  #64 x 128 pixels
#hog.setSVMDetector(cv2.HOGDescriptor_getDefaultPeopleDetector())

#Multi-scale detection and display.
rects_list = []
tracking_ok=0   #represents if the tracking was made ok or not
tracking_error=0    #represents when the tracking failed
epsilon_h=30
epsilon_w=30
d_max=60
#middle_init=[125,100]   #you need to give approximately
#                        #the middle of the face of the person on first frame
rect_obj_list=[]    #contains the rectangles of the tracking in the pictures.


for i in range(0,N):
    im = imlist[i]
    im1 = im
    rects = hogFaceDetector(im, 1)

    if len(rects)==0:
        tracking_error+=1

    else:

```

```
        for faceRect in rects:
            x1 = faceRect.left()
            y1 = faceRect.top()
            x2 = faceRect.right()
            y2 = faceRect.bottom()
            rects_list.append([x1,y1,x2,y2,i])
        #Trace the white rectangle on the picture
        rect1=rects_list[-1]
        cv2.rectangle(imlist[i], (x1, y1), (x2, y2),  255, 2)
        #Tracking :
        #Previous object tracked correctly
        if len(rect_obj_list)==0:
#            obj=[80,100,120,140]        #illumination
#            obj=[203,98,247,141,0]      #obstruction
            obj=[201,34,237,70]         #scale



                                        #you need to give approximately
                                        #the middle of the face of the person on first frame
                                        #give random values at first and then fill it with the
    first element
                                        #of rects_list
        else:
            obj=rect_obj_list[-1]


        #Check if tracking correct
        new_mid = [(x1+x2)/2,(y1+y2)/2]
        prev_mid = [(obj[0]+obj[2])/2,(obj[1]+obj[3])/2]
        if ((abs((obj[3]-obj[1])-(rect1[3]-rect1[1]))<epsilon_h)and(abs((obj[2]-obj[0])-(rect1[2]-
    rect1[0]))<epsilon_w)
                and(abs(new_mid[0]-prev_mid[0])+abs(new_mid[1]-prev_mid[1])<d_max)):
            rect_obj_list.append([rect1[0],rect1[1],rect1[2],rect1[3],i])
            cv2.putText(imlist[i], "Id1", (rect1[0], rect1[1]-5),cv2.FONT_HERSHEY_SIMPLEX, 0.6, 255,
     2)
            tracking_ok+=1
        else:
             tracking_error+=1
```

4.1.2 Parameters

The HOG method doesn't have any parameters. Although, we added some parameters for the tracking part, such as the change in size with both epsilons parameters and the change of distance with the dmax parameter.

4.2 Method 2 : local binary patterns extraction

4.2.1 Implementation

The LBP method is already trained in "Open cv", we imported it using it's XML file. The method is really similar compared to the HOG code, the only thing that really changes is the descriptor and the format of the rectangles.

```
####
## Initialization of the detector
####

#load cascade classifier training file for lbpcascade

lbp= cv2.CascadeClassifier('C:/Users/lucmi\Anaconda2/pkgs/libopencv-3.4.2-h875b8b8_0/Library/etc/
    lbpcascades/lbpcascade_frontalface.xml')

rects_list = []
epsilon_h=30
epsilon_w=30
d_max=200
tracking_ok=0   #represents if the tracking was made ok or not
tracking_error=0    #represents when the tracking failed
rect_obj_list=[]    #contains the rectangles in a picture.
```

```
16
17 for i in range(0,N):
18     im = imlist[i]
19     im1 = im[:, :, 0]
20     rects = lbp.detectMultiScale(im1, scaleFactor=1.1, minNeighbors=5, minSize=(10, 10) )
21     if len(rects)==0:
22         tracking_error+=1
23
24     else:
25         rects_list.append([rects[0,0],rects[0,1],rects[0,2],rects[0,3],i])
26
27 #         for x, y, w, h in rects:
28 #             cv2.rectangle(im, (x, y), (x + w, y + h), (0, 255, 0), 2)
29
30         rect1=rects_list[-1]
31         cv2.rectangle(imlist[i], (rect1[0], rect1[1]), (rect1[0]+rect1[2], rect1[1]+rect1[3]), (255,
      255,255), 2)
32         #Tracking :
33         #Previous object tracked correctly
34         if len(rect_obj_list)==0:
35 #             obj=[99,105,33,33]              #illumination
36 #             obj=[197,93,52,52]              #obstruction
37             obj=[185,73,43,43]              #scale
38
39                                             #you need to give approximately
40                                             #the middle of the face of the person on first frame
41                                             #give random values at first and then fill it with the
    first element
42                                             #of rects_list
43         else:
44             obj=rect_obj_list[-1]
45
46
47         #Check if tracking correct
48         new_mid = [rect1[0]+rect1[2]/2,rect1[1]+rect1[3]/2]
49         prev_mid = [obj[0]+obj[2]/2,obj[1]+obj[3]/2]
50
51         if ((abs(obj[2]-rect1[2])<epsilon_h)and(abs(obj[3]-rect1[3])<epsilon_w)
52                 and(abs(new_mid[0]-prev_mid[0])+abs(new_mid[1]-prev_mid[1])<d_max)):
53             rect_obj_list.append([rect1[0],rect1[1],rect1[2],rect1[3],i])
54             cv2.putText(imlist[i], "Id1", (rect1[0], rect1[1]-5),cv2.FONT_HERSHEY_SIMPLEX, 0.6,
    (255,255,255) , 2)
55             tracking_ok+=1
56         else:
57              tracking_error+=1
```

4.2.2 Parameters

In the usual LBP method you can define a parameter for the distance between the new histogram you get and the histogram of your object. Here, it's already set in the **OpenCV**. Moreover, we added some parameters for the tracking part, already explained in the previous method.

5 Evaluation/Analysis of the results

5.1 Illumination Variation

For the illumination variation case, what we expect is the HOG method to be really precise and the LBP method will find human face when the man is fully in the sun or in the sunshade. For the HOG method we obtain this result :

Here, the label ID on the man's face means the man's face is tracked, the man with a rectangle on his face but without a label ID means we can detect the man's face but we can't track the man at that time.

As we expected, HOG method is really precise when we to detect the man's face, but when the man is at the edge of sun and sunshade, HOG can't track the man's face. The weight of angles in the HOG vector changed and the

imbalance between the weight on the right and the left of the face makes it impossible to detect as a face for HOG. We can see that in the following picture :

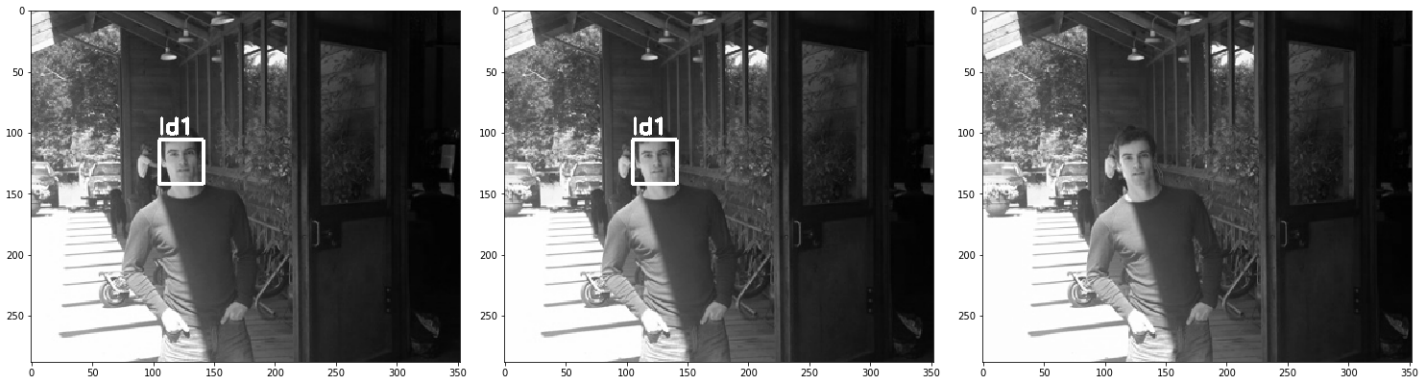

Fig. 5 – Result 1 of the HOG method

We find something else that is interesting, sometimes because of a gap between pictures where HOG detected the face of this man, the next pictures that detect the man's face will not say it's our subject. This simply comes from the fact that we are too far from the last rectangle tracked that represented our subject (the distance criteria). We thought about this problem and thought that we could solve it by increasing the radius of research of the rectangle every new pictures without a rectangle from the method. Unfortunately, we also thought that this could lead to some other mistakes, like starting to track another person in the picture. We are lucky that in our examples, the people move around pretty much in a restrained space, which leads them to coming back near to the previous rectangle, and then we can follow them again. This phenomenon can be explained by the two following pictures :

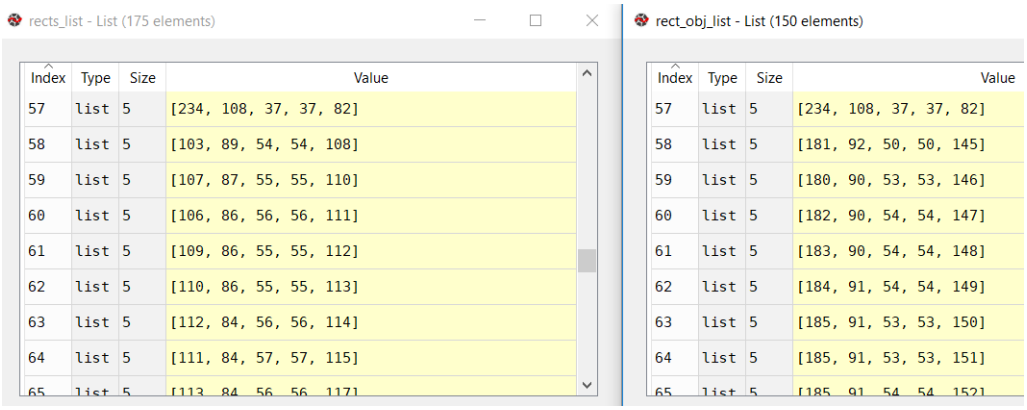

Fig. 6 – Gaps in between the two lists of rectangles

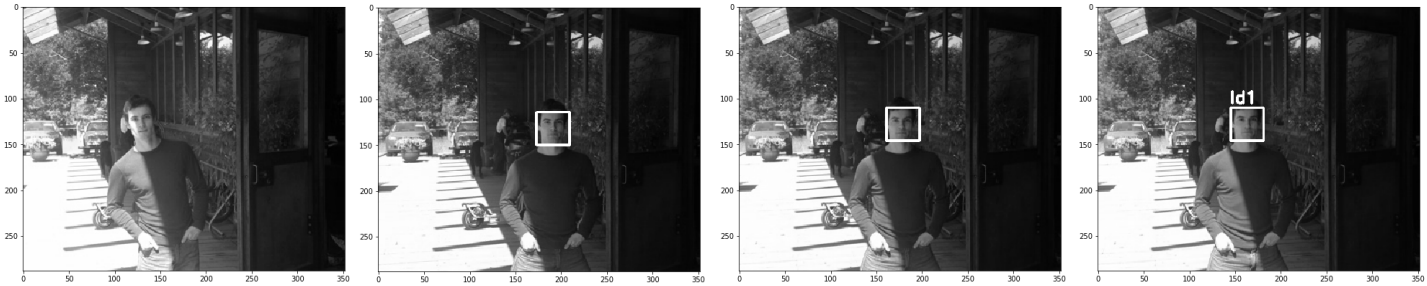

Fig. 7 – Result 2 of the HOG method

For LBP results, we can see that the man is tracked by LBP when the man is fully in the sun, when the man is between the sun and sunshade, we can't detect the man's face. That is because LBP needs it's neighbour's pixels to create a binary code, the pixel values changed a lot when the man is between sun and sunshade. What we expected is that the man's face can be tracked when the man is fully in the sunshade or in the sunlight, we can prove it on the result 1 and 2 of LBP.

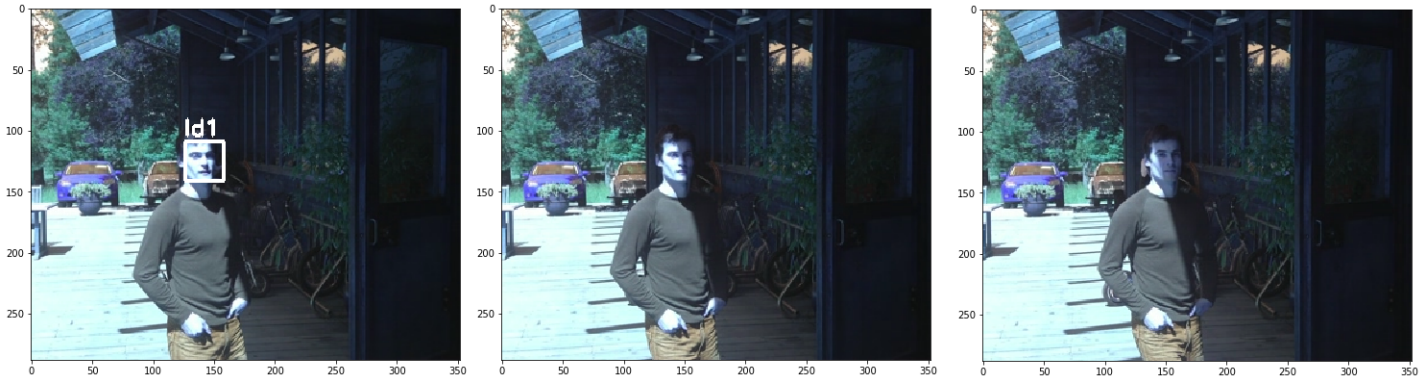

Fig. 8 – Result 1 of the LBP method

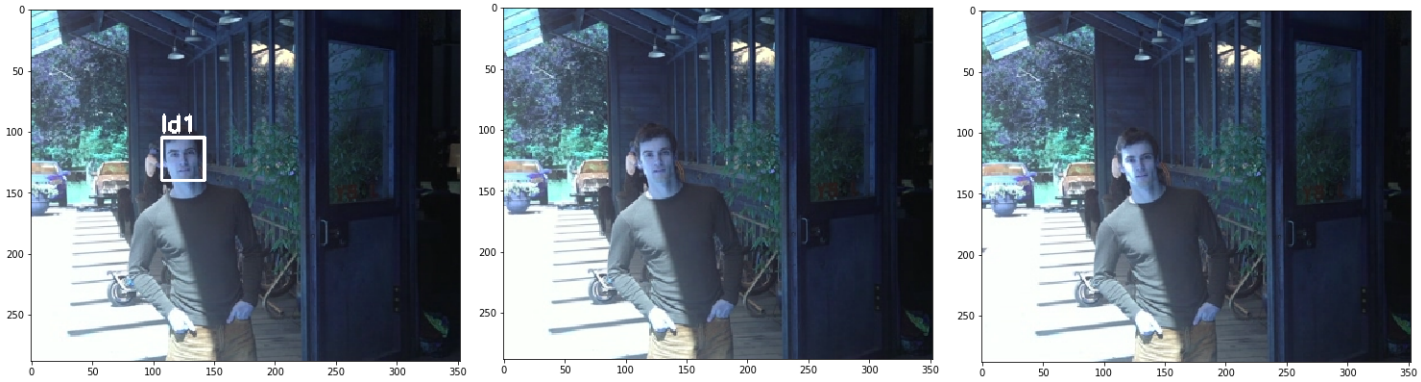

Fig. 9 – Result 2 of the LBP method

Now, we also have the problem of Gaps in between the rectangles list while using LBP. This problem can be evaluated using the **tracking_obj_detected** variable in the evaluation criteria, we can see this problem in the next picture :

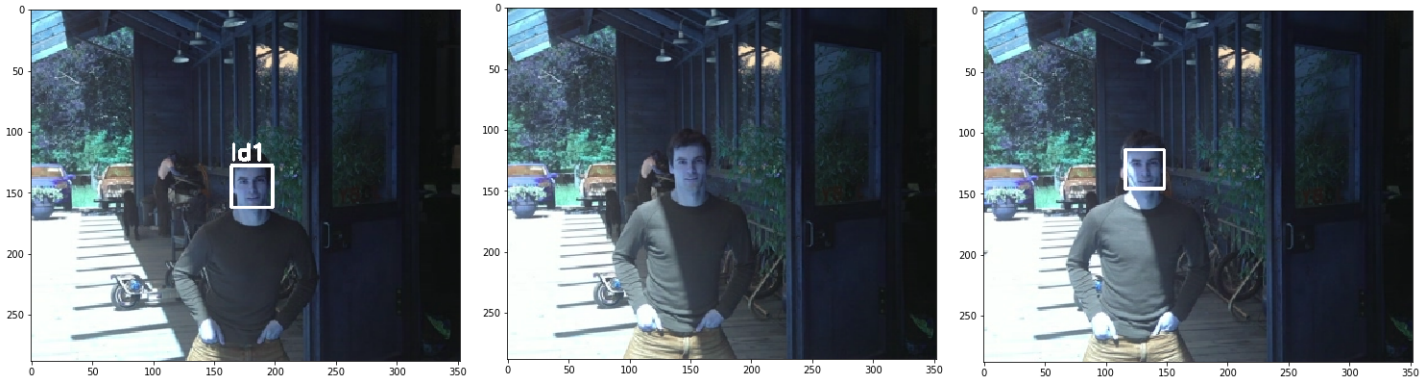

Fig. 10 – Result 3 of the LBP method

Now, we want to compare the performance of both the techniques, for that we use our evaluation criteria and we can say that from the tables below, we can see that the HOG performance is much better than LBP.

rect_obj_list	list	136	[[81, 106, 117, 142, 0],
rects_list	list	143	[[81, 106, 117, 142, 0],
tracking_complete	float	1	0.7906976744186046
tracking_error	int	1	36
tracking_obj_detected	float	1	0.951048951048951
tracking_ok	int	1	136
tracking_uncomplete	float	1	0.20930232558139536

Fig. 11 – Illumination criteria of the HOG method

tracking_complete	float	1	0.22674418604651161
tracking_error	int	1	133
tracking_obj_detected	float	1	0.48148148148148145
tracking_ok	int	1	39
tracking_uncomplete	float	1	0.7732558139534884

Fig. 12 – Illumination criteria of the LBP method

For HOG the object detected by HOG was tracked by our method in more than 95% of cases which means the gaps were not too abundant compared to LBP where the gaps caused only 48% of the object detected to be tracked. The uncomplete tracking object is only 20% in HOG compared to 77% in LBP which means that more than 3/4th of the pictures were not tracked. Finally, the complete tracking is of 79% using HOG compared to 22% using LBP. From all of those results we can say that hog is much better than LBP in the illumination change case.

5.2 Occlusion

What surprised us is that HOG performed better than we expected. The HOG can track the man's face when the man occluded part of his face, more precisely, the man's face when the man occluded the his forehead. I am wondering that the HOG in the Dlib is trained by the human's eyes, nose and mouth(the human's distinctive features). As we can see that when the man occluded one of his eyes, the HOG didn't work anymore. Here, we can say that when the man occluded part of his face, but if we still can see the man's eyes, nose, mouth, the man can be tracked by HOG method.

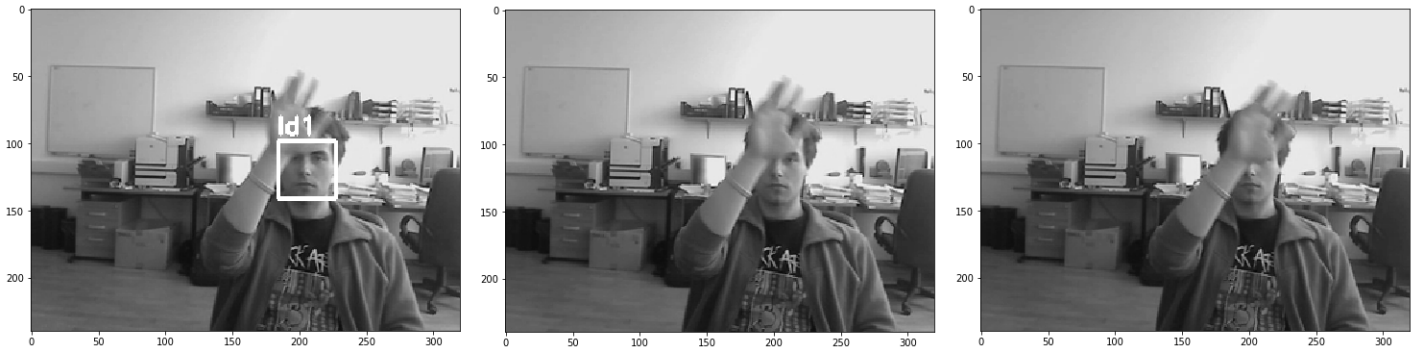

Fig. 13 – Result 1 of the HOG method

Also once again we can see that there's the Gap problem we presented before.

rects_list - List (189 elements)

Index	Type	Size	Value
54	list	5	[155, 98, 199, 141, 66]
55	list	5	[177, 102, 213, 138, 73]
56	list	5	[179, 98, 223, 141, 74]
57	list	5	[179, 98, 223, 141, 75]
58	list	5	[184, 98, 227, 141, 76]
59	list	5	[107, 93, 151, 137, 106]
60	list	5	[107, 93, 151, 137, 108]
61	list	5	[112, 93, 155, 137, 110]

rect_obj_list - List (173 elements)

Index	Type	Size	Value
54	list	5	[155, 98, 199, 141, 66]
55	list	5	[177, 102, 213, 138, 73]
56	list	5	[179, 98, 223, 141, 74]
57	list	5	[179, 98, 223, 141, 75]
58	list	5	[184, 98, 227, 141, 76]
59	list	5	[146, 98, 189, 141, 130]
60	list	5	[146, 98, 189, 141, 131]
61	list	5	[153, 102, 189, 138, 132]

Fig. 14 – Result 2 of the HOG method

For LBP, we guessed that the man can be tracked when the man is occluded by his hand sometimes, because the hand texture is similar with the face texture. Unfortunately, we can't track the man's face when the hand is occluded one of his eyes and LBP also tracked the wrong object once. This thing proves that our algorithm may be sensitive to tracking the wrong things sometimes.

Fig. 15 – Result 1 of the LBP method

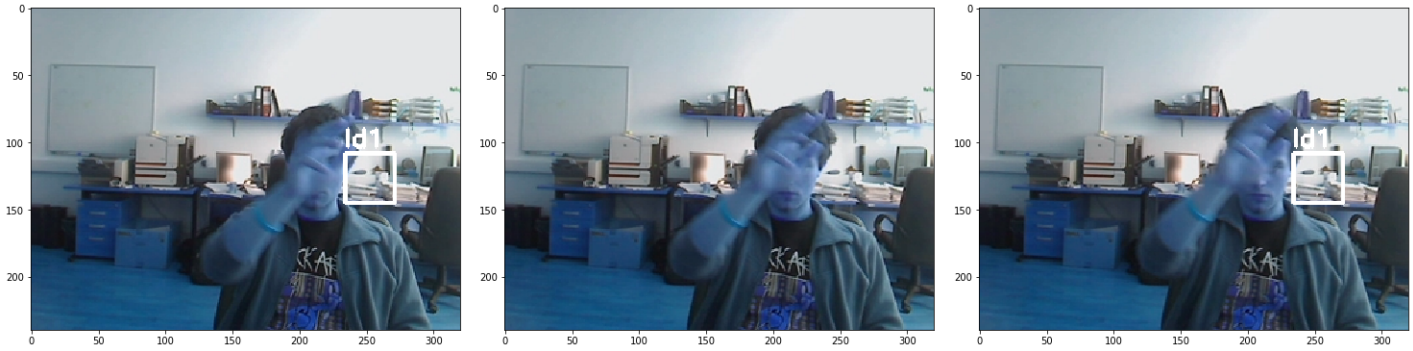

Fig. 16 – Result 2 of the LBP method

Also, once again we can see the gap problem.

Fig. 17 – Result 3 of the LBP method

Now, we want to compare the performance of both the techniques, for that we use our evaluation criteria and we can say that from the tables below, we can see that the HOG performance is a bit better than LBP.

Name	Type	Size	Value
rect_obj_list	list	173	[[203, 98, 247, 141, 0],
rects_list	list	189	[[203, 98, 247, 141, 0],
tracking_complete	float	1	0.6479400749063671
tracking_error	int	1	94
tracking_obj_detected	float	1	0.9153439153439153
tracking_ok	int	1	173
tracking_uncomplete	float	1	0.352059925093633

Fig. 18 – Occlusion criteria of the HOG method

Name	Type	Size	Value
rect_obj_list	list	150	[[197, 93, 52, 52, 0],
rects	int32	(2, 4)	ndarray object of numpy module
rects_list	list	175	[[197, 93, 52, 52, 0],
tracking_complete	float	1	0.5617977528089888
tracking_error	int	1	117
tracking_obj_detected	float	1	0.8571428571428571
tracking_ok	int	1	150
tracking_uncomplete	float	1	0.43820224719101125

Fig. 19 – Occlusion criteria of the LBP method

For HOG the object detected by HOG was tracked by our method in 91% of cases which means the gaps were a bit less compared to LBP where the gaps caused only 85% of the object detected to be tracked. The uncomplete tracking object is only 35% in HOG compared to 43% in LBP which means that less than half of the pictures were not tracked. Finally, the complete tracking is of 64% using HOG compared to 56% using LBP. From all of those results we can say that HOG is a bit better than LBP in the occlusion case.

5.3 Scale Changes

Unfortunately, both of the methods worked not good, the reason we guessed is that the woman changed the place she stood really fast. Therefore, this data-set is not good for the scale changes problem, because this data-set also has the object fast moving problem. We tested other data-sets for the scale changes problem too, but in all of the data-set human moved really fast and the human face is really small.

The woman became smaller when she changed place, HOG can track her even when her face is a bit blurry in the result 2. As we said above, the woman moved so fast that HOG cannot track it on time sometimes, that affected the results.

Fig. 20 – Result 1 of the HOG method

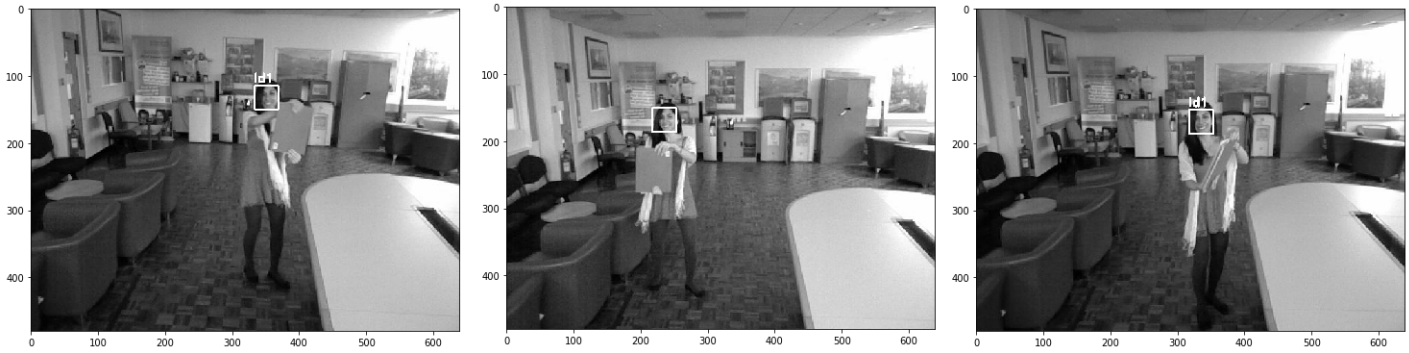

Fig. 21 – Result 2 of the HOG method

For LBP method, the woman can be tracked when she moved, but when her face is blurry at the result 2, LBP can't track her face. The pixels changed a lot when the woman become small and big suddenly, LBP used really different neighbour's pixels and can't track her face anymore. Also we tracked something that was not her face.

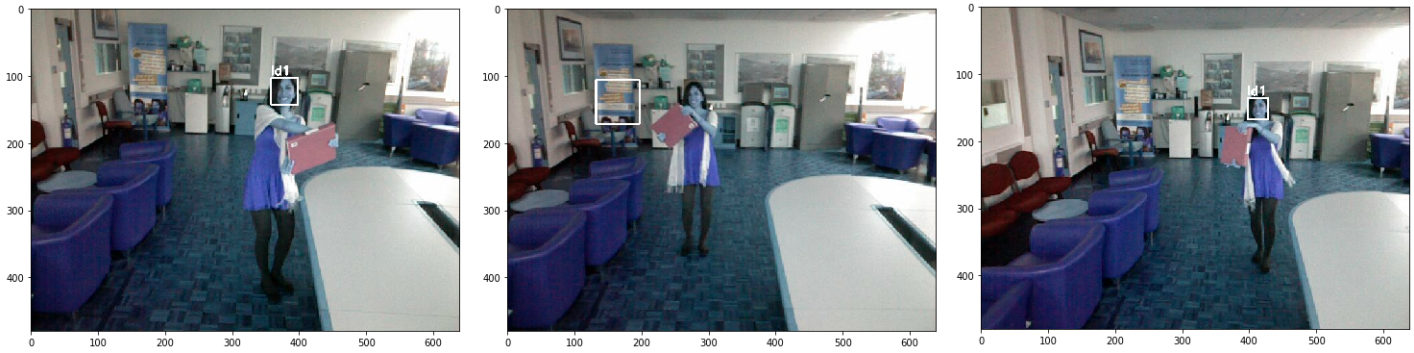

Fig. 22 – Result 1 of the LBP method

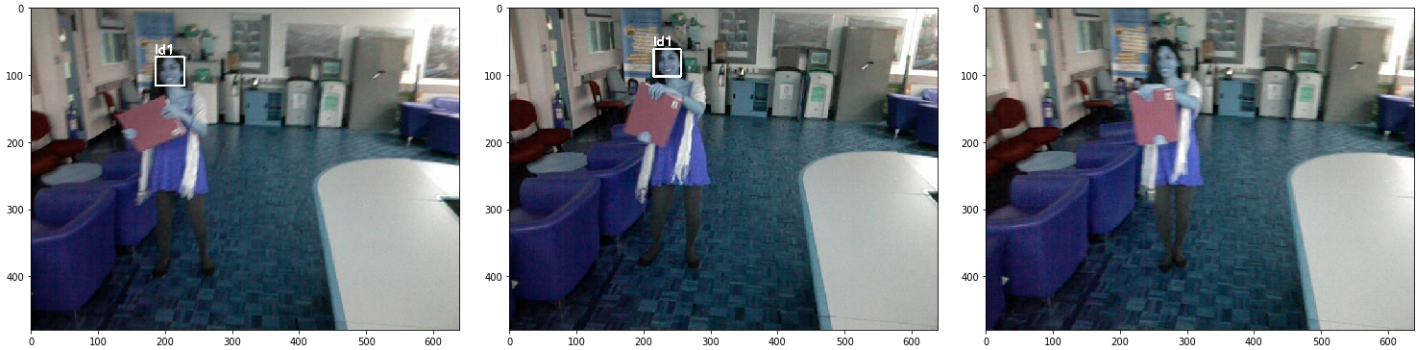

Fig. 23 – Result 2 of the LBP method

We also found that the blurriness didn't affect the detection of the HOG method. We can see that in the next picture :

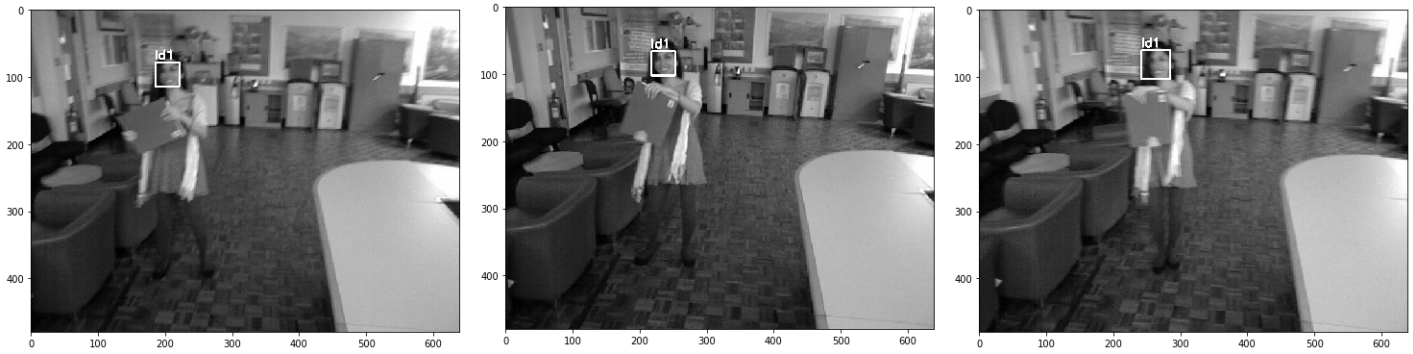

Fig. 24 – Result 3 of the HOG method

Now, we want to compare the performance of both the techniques, for that we use our evaluation criteria and we can say that from the tables below, we can see that the HOG performance is much better than LBP.

rect_obj_list	list	56	[[201, 34, 237, 70, 2], [18…
rects_list	list	56	[[201, 34, 237, 70, 2], [18…
tracking_complete	float	1	0.32
tracking_error	int	1	119
tracking_obj_detected	float	1	1.0
tracking_ok	int	1	56
tracking_uncomplete	float	1	0.68

Fig. 25 – Scale criteria of the HOG method

rect_obj_list	list	8	[[185, 73, 43, 43, 10], …
rects	tuple	0	()
rects_list	list	13	[[185, 73, 43, 43, 10], …
tracking_complete	float	1	0.045714285714285714
tracking_error	int	1	167
tracking_obj_detected	float	1	0.6153846153846154
tracking_ok	int	1	8
tracking_uncomplete	float	1	0.9542857142857143

Fig. 26 – Scale criteria of the LBP method

For HOG the object detected by HOG was tracked by our method in 100% of cases which means there is no gaps for HOG compared to LBP where the gaps caused 61% of the object detected to be tracked. The uncomplete tracking object is 68% in HOG compared to 95% in LBP which means that most of the pictures were not tracked on both methods, expecially for LBP. Finally, the complete tracking is of 32% using HOG compared to only 4% by LBP. From all of those results we can say that hog is much better than LBP in the different scales case.

6 Conclusion

In conclusion we can say that in most of the exemples here the HOG is better than the LBP, this fact may come from the fact that the **dlib** face recognition HOG is really well trained. In fact, when we were using the HOG people detection in the begining of this lab we found that the results were not that good. Unfortunately we couldn't prove all of our points, because of some tracking problems. Maybe we could have done a better tracking algorithm be we decided not to change it because we didn't want to track some inaccurate objects.

7 Code Library and Parts used

we used opencv 3.4.1 to do most of our things in the code. The code is wrote by ourselves for the most part. We used the code on "Stack Overflow" to import pictures on a forum :

https ://stackoverflow.com/questions/30230592/loading-all-images-using-imread-from-a-given-folder

This forum helped us understand how to import pictures from a file.

We used the code to display a label on the pictures, the website link is as follows :

https ://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/

We used the website below that said we can use Dlib for HOG face detection, then we used the code about how to create rectangles of human face in Dlib library.

https ://www.learnopencv.com/face-detection-opencv-dlib-and-deep-learning-c-python/

The rest of the code we wrote it by ourselves.