Application of Artificial Intelligence Algorithms for Image Processing

Nataliya Boyko[0000-0002-6962-9363], Andriy Bronetskyi[0000-0003-2415-6723],

Nataliya Shakhovska[0000-0002-6875-8534]

Lviv Polytechnic National University, Lviv79013, Ukraine
nataliya.i.boyko@lpnu.ua, nataliya.b.shakhovska@lpnu.ua

Abstract. This article describes the approach to machine learning to identify objects that are capable of handling images extremely quickly and reaches high detection rates. This paper describes three main areas for research on image processing. First, it is the introduction of a new image called "Integral image", which allows you to quickly calculate the functions that our detector uses. The second is the learning algorithm based on AdaBoost, which selects a small number of critical visual functions from a larger set and provides extremely efficient classifiers [6]. The third installment is a method of combining increasingly complex classifiers in the "cascade", which allows background areas to quickly reject the image, spending more calculations on promising object-like areas. The cascade can be considered as a specific mechanism of focusing attention. In the face detection system, the system displays detection rates comparable to the best previous systems. This paper describes the process of face-recognition using OpenCV library in Python.

Keywords: OpenCV, algorithm, frame, cascade.

1 Introduction

Computer vision is a fast-growing field dedicated to analyzing, modifying, and high-level understanding of images. Its objective is to determine what is happening in front of a camera and use that understanding to control a computer or robotic system [10].

OpenCV is a cross-platform, free for use library mainly aimed at real-time computer vision. Officially, it was developed by Intel and launched in 1999. It is written in C++ and its primary interface is in C++, but there are also buildings in Python and Java. OpenCV runs on the following desktop operation systems: Windows, Linux, macOS. It also runs on the following mobile operation systems: Android, iOS, BlackBerry. So, as you can see, you can use it for you projects for free, and on all the operation systems you need [8].

The object of study is to investigate image recognition algorithms.

The subject of study is the OpenCV library and a opportunities of this library to recognize Faces.

The purpose of the work is to create a program that will recognize faces and to learn about how it works.

2 Setting the task

Formulation of the problem: learn about:

1. Viola-Jones Algorithm
2. Haar-like features
3. Integral Image
4. Adaptive boosting.

And finally making a Face-Recognition program.

3 Materials and methods

The procedure proposed by the authors to identify objects classifies the image based on the value of simple functions. The main reason is that functions can facilitate the encoding of special domain knowledge that is difficult to learn using a small amount of training data. For this system, there is also a second critical motivation for functions: the function-based system works much faster than the system based on pixels.

The simple functions used in this paper are basic Haar functions [10].

The task of detecting faces is as follows:

- the image of *KxM* pixels is given;
- it is necessary to find the coordinates of the rectangles of the minimal size, containing the faces of this image;
- all other things must be ignored, including buildings, trees and other parts of the body.

3.1. Mathematical statement of the problem of face detection

- Ω is a space of images, $\omega \in \Omega$ is an image;
- $M = \{1, 2, \ldots N\}$ - set of classes indexes;
- $\Omega_m \subset \Omega$, $m = 1; N$, $\Omega_i \cap \Omega_j = \emptyset$ for $i \neq j$ and $\Omega_m \times N\, m=1 = \Omega$; $g : \Omega \rightarrow M$ is an indicator function that is unknown;
- X - space of signs;
- $x: \Omega \rightarrow X$ - a function that sets the image ω as its vector of signs $x \times (\omega)$;
- $Ks \subset X$, $s = 1; N$, $Ki \cap Kj = \emptyset$ for $i \neq j$ and $K_m \times N\, m=1 = X$; $G: X \rightarrow M$ is the deciding rule that converts the number of the class to which it belongs, in accordance with the character vector of the image.

Task of classification with a teacher: based on a plurality of precedents (g_j, x_j), $j = 1; N$ (training sample) it is necessary to construct a rule of G rule that minimizes the number of errors [7].

There are many methods for detecting faces, among which the best performance and quality of work is considered to be a cascade detector based on Haar-like features, first introduced by Viola and Jones.

The Viola-Jones method includes [8-9]:

- The Viola-Jones algorithm is based on Haar-like features;
- Integral Image (summed-area tables) is used to calculate the sign;
- One of the advantages of the method is its independence from scale;
- There are many modifications of these methods proposed by Lee et al.

Haar-like features for face detection are the identifying features of an image. All Haar-like features have the shape of a rectangle: the sum of the values of pixels in the same sector is subtracted from the sum of pixels in the other sector of the image (1).

$$V = \frac{w_1 * \sum_{i \in area1} i(x,y) + w_2 * \sum_{i \in area2} i(x,y) + w_3 * \sum_{i \in area3} i(x,y)}{S^2} \quad (1)$$

where w_i is the constants that are inversely proportional to the size of the corresponding region (the number of pixels); s is a scaling factor ($s \geq 1$); w_i have different signs in white and black areas respectively [2-5]

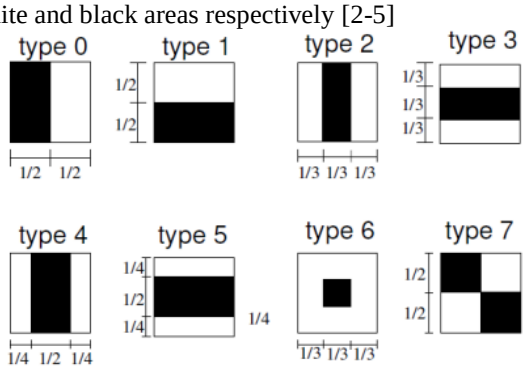


Fig. 1. Signs of Haar

Linhart and Meyde (2002) added an additional set of features to the proposed Viola-Jones algorithm. These features improved the accuracy of the method, but required a pre-date learning time [6-8].

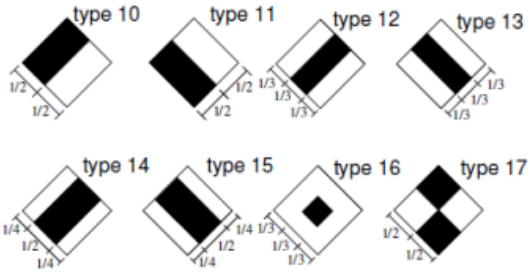


Fig. 2. Signs of Haar added by Linhart and Maydah

The set of features presented in the previous matrix is superfluous and over-represented the object, since their number is larger than the number of pixels in the image. In order to achieve the invariance of the scale, it is enough to divide the value

of the sign into the area (measured in pixels). These features are computed on various scales, using the Integral Image representation method, also known as SAM (Summed-area tables).

SAT is defined as a matrix in which each element contains the sum of all pixels lying in the top left-hand corner of the image with the vertex in the current pixel. SAT I (x, y) will count as [1-6]:

$$I(x, y) = \sum_{x \le x_i, y \le y_i} i(x_i, y_i) \quad (2)$$

where I(x_i, y_i) is a set of image pixels..

The sum of pixels in any rectangular area of an image can be calculated in 4 steps:

$$\sum pix = pt4 - pt3 + pt2 + pt1 \quad (3)$$

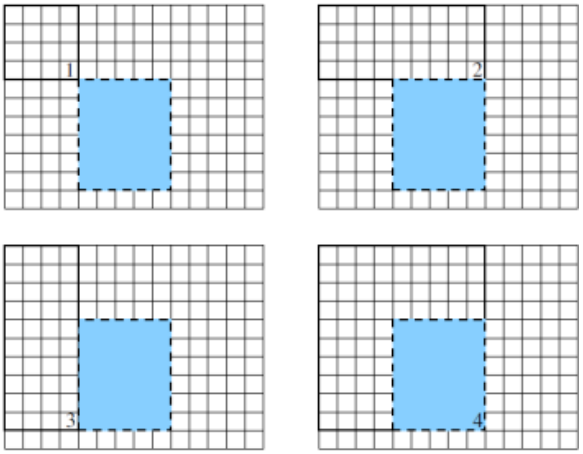


Fig. 3. Example SAT

3.2. Stages of constructing a classifier

The process of constructing a classifier includes three stages, presented in Fig. 4

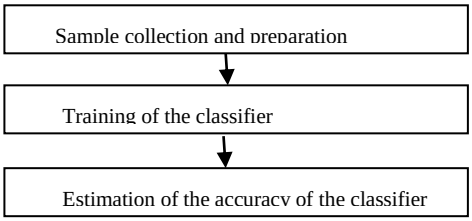


Fig. 4. Stages of constructing a classifier

The process of detecting faces by a classifier takes place as follows:

- Detection of faces occurs using the method of a scanning window;

- Each iteration selects a certain size of age-by which the detector passes through the image and defines the area;
- Haar-Like fetures are calculated for each area;
- If the values correspond to the values in the classifier, then this window will be considered as the face.

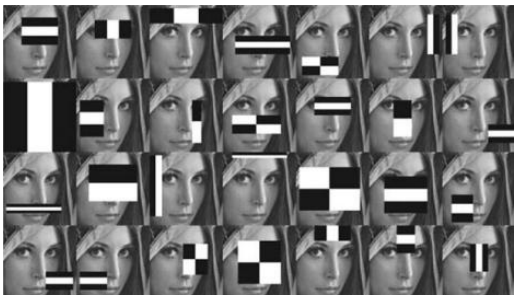


Fig. 5. Examples of calculating Haar signs

The accuracy of the classifier is determined in the following way:
Known main metrics:

$$precision = \frac{TP}{TP+FP}$$
$$recall = \frac{TP}{TP+FN} \quad (4)$$
$$accuracy = \frac{TP+TN}{TP+FP+TN+FN}$$

TP (True Positives) - the number of correctly classified samples from the positive sample (so-called truth-positive cases);

TN (True Negatives) - correctly classified non-negative images (true non-negative cases);

FN (False Negatives) - the number of positive pri-systems classified as negative (I-type error) (false negatives);

FP (False Positives) - number of negative examples classified as positive (type II error) (false positive cases).

Here are some relative indicators for ROC analysis:

$$TPR = \frac{TP}{TP+FN}$$
$$FPR = \frac{FP}{TN+FP} \quad (5)$$
$$S_e = 1 - TPR - \text{sensitivity}$$
$$S_p = 1 - FPR - \text{specificity}$$

AUC (Area Under Curve) is the area under the curve.

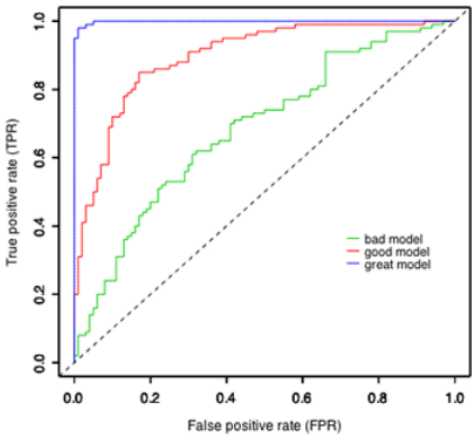


Fig. 6. An example of ROC curves

4 Experiments

Viola-Jones Algorithm is the algorithm that lies in the foundation of OpenCV library. And it is one of the most potential and powerful tools for REAL-TIME CV. It was developed in 2001 by Paul Viola and Michael Jones. This algorithm consists of 2 stages: Training and Detection. We will use a prepared cascade (training model) for detection in this article. So, the first step of Viola-Jones Algorithm is turning the given frame into gray-scale. This step is necessary because it will decrease a lot of work that computer would do and will change our colors in the interim of 0 and 1 that will help us a lot in our next steps. But don't worry about that, out picture will be RGB at the end of detection again [10]. So, the next step of this algorithm is a little square that starts from the top left corner and moves to the right. So, the all next steps are occurring in this square. Further this algorithm is looking for the face. But how is it looking for the face we will discuss later. But for now, we will remember that it is looking for certain features on the face, and for features we are going to mean that it is looking for the eyebrows, nose, eyes and other things on our face. So, let`s take a look on this picture.

Fig. 7. First step of Viola-Jones algorithm

Computer has detected this man`s ear and it says: “Ok, this can be the face” but it realizes that for it to be a face has to be an eye, mouse, nose, etc...So, it keeps moving.

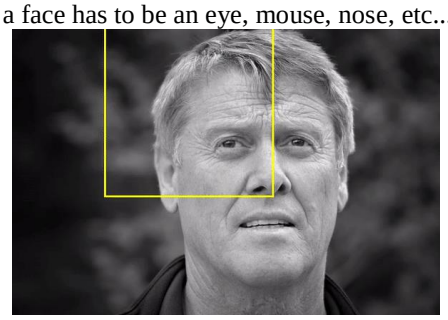


Fig. 8. Some step of Viola-Jones algorithm

In this position computer detect an ear and eye, but it is not the full face still.

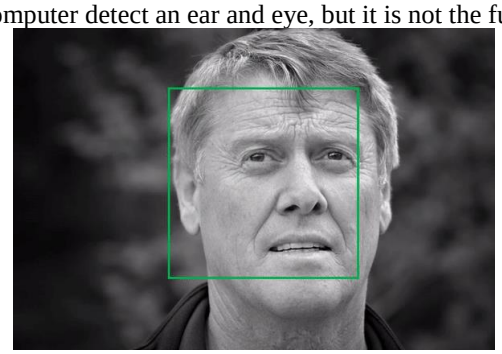


Fig. 9. Detected face in square

After the first line, our square fall down a little bit and kept moving to the right. And in this position, computer can see all the face, and it can clearly say that it is a face.

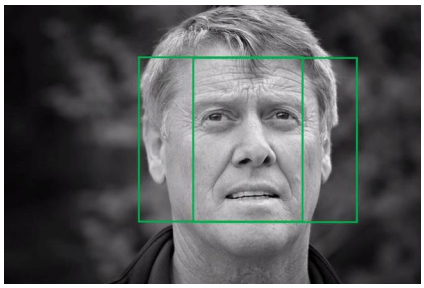


Fig. 10. Detected face in another square

In the next position, it can see all the face too, and it can make a decision that face is right here. But it keeps moving till the end of the frame because there can be other faces too.

So that is in nutshell how this algorithm works. And we will continue

Our next step is Haar-like features. When we talked about detecting square, I said that we will discuss it later. And it is the moment to learn what is going on in this

square. So, what is actually Haar-like features?Haar-like features are digital image features used in object recognition. They are the edge features, the line features and four-rectangle features.

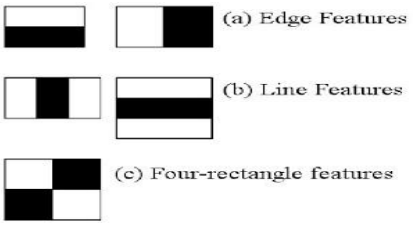


Fig. 11. Haar-like features

So, what does it mean to be a feature? On our frame we have basically pixels. And somewhere on this frame we can have for example like an edge feature where one part is much darker or brighter than the other one. Photo example:

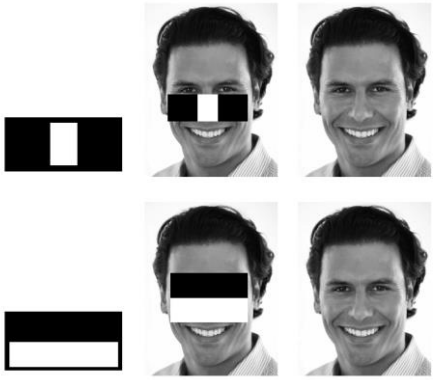


Fig. 12. Examples of Haar-like features

This is one of the reasons why we turned our frame into gray-scale. And so, on only 1 frame you can have a millions of these features, and I can`t even imagine what computer we would need to process all this features on the frame in real time. But we have our savior Adaptive boosting, and we will talk about it soon.

And now let`s talk about how these features are identify. Let`s take a look on this edge feature.

On the frame, this feature is made by gray-scale pixels, so these pixels are marked from 0 to 1 on the photo and from 0 to 255 in the program code(from white to black). And on this step, Viola-Jones algorithm will compare the AVERAGE number (it is important to remember because we will talk about it later) of white side and black side of the feature. This operation is actually executed on all the frame, that's why the count of these featuresis so big. And so, our algorithm will find all the satisfying features for us, what means that the comparing operation will return the value that is bigger or the same with the value that our trained cascade deal with. Yes, our trained

cascade has even the average position and value for all the Haar-Like features on the face or eyes or everything that this cascade is made for.

In Viola-Jones algorithm the frame that we are working with decomposes to a list with a lot of values from 0 to 255. And how hard it would be to find an average value for even 1 feature? It will take a lot of time what is impossible in real-time recognition.

An integral representation of an image is a matrix that coincides in size with the original image. Each element of it stores the sum of the brightness of all pixels that are left and above this element. Elements of the matrix are calculated according to the following formula:

$$L(x, y) = \sum_{i=0, j=0}^{i \le x, j \le y} I(i, j) \quad (6)$$

where I (i, j) is the pixel brightness of the input image.

Each element of the matrix L [x, y] is the sum of the brightness of the pixels in the rectangle from (0, 0) to (x, y), that is, the value of each pixel (x, y) in integral form is equal to the sum of the values of all pixels of the original image left and above the given pixel (x, y). The calculation of the matrix takes line-time, proportional to the number of pixels in the image, so the integral image is calculated in one pass.

The calculation of the matrix is possible by the formula 7:

$$L\,(x, y) = I\,(x, y) - L\,(x - 1, y - 1) + L\,(x, y - 1) + L\,(x - 1, y), \quad (7)$$

where the designation corresponds to formula 6.

With such an integral matrix you can quickly calculate the sum of the pixels of any arbitrary square, arbitrary square.

Let us be interested in the sum of pixels in the rectangle ABCD:

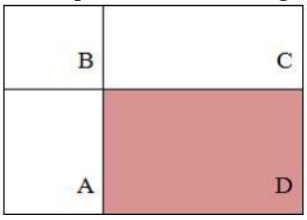


Fig. 13. The clear that the sum of the brightness of the pixels

From Fig.13 it is clear that the sum of the brightness of the pixels of the all-red face of the rectangle can be expressed in terms of the sum and the difference of adjacent rectangles by the following formula:

$$S\,(A\,A) = L\,(Ax, Ay) + L\,(Cx, Cy) - L\,(Bx, By) - L\,(Dx, Dy) \quad (8)$$

where *A, B, C, D* are points, and A_x, A_y, etc. - these are their co-ordinates.

Integral image solves this problem. So, let`s look on this image:

1	2	5	7	2	8	0	6	4	6
9	8	0	4	9	5	10	7	10	3
7	6	10	2	0	10	4	9	10	8
3	8	1	5	4	8	0	9	5	8
9	5	0	1	3	4	1	9	6	1
1	2	5	6	9	9	0	2	4	0
1	2	4	1	6	6	10	4	2	5
5	6	2	10	5	3	9	10	10	2

Fig. 14. Example of Frame-matrix

Let`s imagine that this matrix is our frame, and our gray-scale is from 0 to 10. We need to find a value of edge feature in this place

1	2	5	7	2	8	0	6	4	6
9	8	0	4	9	5	10	7	10	3
7	6	10	2	0	10	4	9	10	8
3	8	1	5	4	8	0	9	5	8
9	5	0	1	3	4	1	9	6	1
1	2	5	6	9	9	0	2	4	0
1	2	4	1	6	6	10	4	2	5
5	6	2	10	5	3	9	10	10	2

Fig. 15. Some place in integral image

So we will add all these numbers and divide on the count of numbers. It will not take a lot of time, but as I said there is a lot of these features, and our image will be a lot bigger that this one. So what Integral Image actually do?

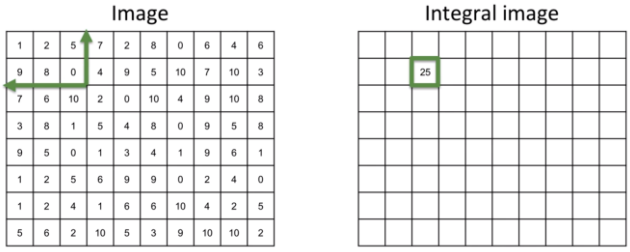


Fig. 16. Calculations in integral image

As we can see on the photo, it creates a new matrix, with the main idea in that every square will be the sum of the values of the original image from above and to the left.

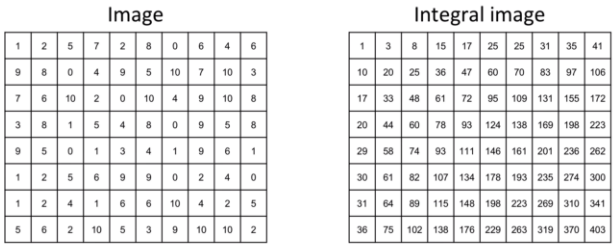


Fig. 17. Example of converted image

And that is how Integral image works. But how it helps?

It shorts the count of our operations to maximum 4 of them. For example, if we need to find the average value in this square:

Fig. 18. Place to calculate

we can just perform 4 operations:

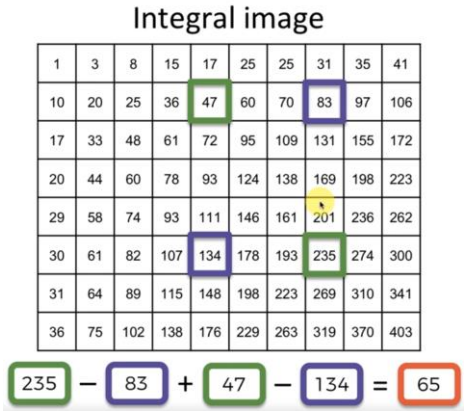


Fig. 19. Operations to calculate Integral Image and our average value will be found.

4.1. Adaptive Boosting

Adaptive Boosting is a complex of methods that increase the accuracy of analytical models. An effective model that assumes few classification errors is called

"strong". "Weak", by contrast, does not allow reliable separation of classes or accurate predictions, makes a lot of errors in the work. Therefore, boosting (from English Boosting - increase, gain, improvement) means literally "strengthening" the "weak" models [6] - this is a procedure for sequentially constructing a compilation of machine learning algorithms, when each subsequent algorithm seeks to offset the disadvantages of the composition of all previous algorithms.

Before we will start learning what is Adaboost, I must tell a little bit about training Viola-Jones algorithm. Now we know the main phases of how this algorithm works, so we can understand what is training itself. When we want to teach Viola-Jones algorithm to classify for example apples, we put a lot of apple images in it and say that it is apples. It finds a lot of Haar-like features in that images and compare it to each other. But we also must put into our program a non-apple images to compare what IS NOT apple. And our output will be an XML file which is called cascade. We will not make it by ourselves, but you can have a lot of this pre-trained cascades in internet.

Now we can talk about Adaboost. As I said, there is one problem in Haar-Like features. We can have a millions of it in our frame. And even if we will smooth this frame into 24 by 24 pixels, we will have over 180000+ of these features. And during the training it would be very hard because you not only have to check all these features on 1 apple photo, you need to check them on a thousands of it plus there is also a huge count of non-apple photos. And it is where Adaboost can help. With the help of it we are going to take our feature and put them into a classifier. And when you compare features on your apple photos, only 1 feature is very weak on its own. But when we have a classifier of these features, our training will be much more effective and better for later detection. So, when you compare only one feature, it is called weak classifier, but when you compare a lot of it, it is called a strong classifier.

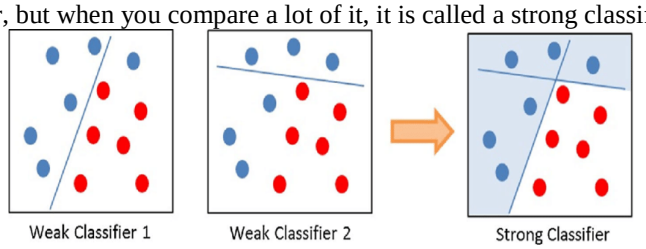


Fig. 20. It is called a strong classifier

4.2. Making a strong classifier

Bousting is an greedy algorithm for constructing a composition of algorithms (greedy algorithm) - an algorithm that at each step makes locally the best choice in the hope that the final solution will be optimal. Busting over selection trees is considered one of the most effective methods in terms of the quality of the classification.

In many experiments, there was an almost unlimited decrease in the frequency of errors on an independent test sample when increasing the composition.

Moreover, the quality of the test sample often continued to improve even after achieving the unambiguous recognition of the entire training sample [7].

Nowadays, the approach of enhancement of simple classifiers is a popular and probably the most effective classification method due to the high speed and efficiency of work and the relative simplicity of implementation.

The boosting algorithm for face search is as follows:

1. Definition of weak classifiers with rectangular signs;

2. For each move of the scanning window, a rectangular sign is calculated on each example;

3. Selects the most appropriate threshold for each sign;

4. Selected the best signs and the best corresponding threshold;

5. A recapture of the sample.

The cascading model of strong classifiers is essentially the same decision tree, where each node of a tree is constructed in such a way as to detect almost all of the images that interest us and to reject regions that are not images. In addition, the nodes of the tree are arranged in such a way that the closer the knot is to the root of the tree, the smaller the number of primitives it consists of and thus requires less time to make a decision. This kind of cascading model is well suited for image processing, in which the total number of detected images is small.

For now, I think that we are ready to make a Face Recognition program.

Our first step will be finding cascades for eye and frontal face. I have found it in this repository https://github.com/opencv/opencv/tree/master/data/haarcascades.

Our program we will make on Python, so let`s get started

Firstly, we will import cv2 library and announce our variables with cascades.

```
import cv2

face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml') # We load the ca
```

Then we will make a function that will be detecting a face and eyes.

```
def detect(gray, frame): # We create a function that takes as input the image
    faces = face_cascade.detectMultiScale(gray, 1.3, 5) # We apply the detect
    for (x, y, w, h) in faces: # For each detected face:
        cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0), 2) # We paint
        roi_gray = gray[y:y+h, x:x+w] # We get the region of interest in the
        roi_color = frame[y:y+h, x:x+w] # We get the region of interest in th
        eyes = eye_cascade.detectMultiScale(roi_gray, 1.1, 3) # We apply the
        for (ex, ey, ew, eh) in eyes: # For each detected eye:
            cv2.rectangle(roi_color,(ex, ey),(ex+ew, ey+eh), (0, 255, 0), 2)
    return frame # We return the image with the detector rectangles.
```

Then we will start our webcam and will make a cycle that will create a window and display our finished frames

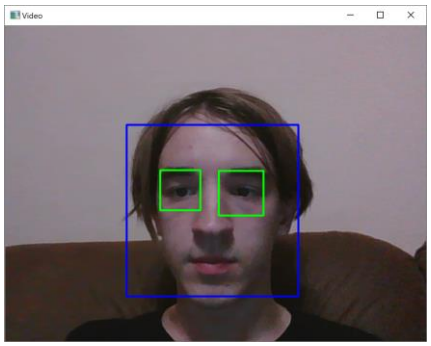
```

video_capture = cv2.VideoCapture(0) # We turn the w

while True: # We repeat infinitely (until break):
    _, frame = video_capture.read() # We get the la
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    canvas = detect(gray, frame) # We get the outpu
    cv2.imshow('Video', canvas) # We display the ou
    if cv2.waitKey(1) & 0xFF == ord('q'): # If we t
        break # We stop the loop.

video_capture.release() # We turn the webcam off.
cv2.destroyAllWindows()# We destroy all the windows
```

In the end, our program works properly



5 Results

Our experiments will be based on the quality of Face Recognition in different situation related on light, distance and moving objects. Firstly, we have to create a universal quality recognition meter, for this we will change our program a little bit. We will meter our recognition quality by dividing count of all the frames that our program returned on the count of all the frames when our program recognized face.

We will add a variable called frames for counting all the frames and a variable called detected for counting all the frames when faces were detected

```
# Detecting success calculate
frames = 1
detected = 1
```

After that we will add one to frame after each loop our program will done and add one to detected when the program recognized face.

```
def detect(gray, frame):
    global detected
    faces = face_cascade.detectMultiScale(gray, 1.3, 5)
    for (x, y, w, h) in faces:
        cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0), 2)
        roi_gray = gray[y:y+h, x:x+w]
        roi_color = frame[y:y+h, x:x+w]
        detected += 1
        eyes = eye_cascade.detectMultiScale(roi_gray, 1.1,22)
        cv2.putText(frame,'Face',(x,y-7), font, 1,(255,255,255),1,cv2.LINE_AA)
        for (ex, ey, ew, eh) in eyes:
            cv2.rectangle(roi_color, (ex, ey), (ex+ew, ey+eh), (0, 255, 0), 2)
    return frame
```

```
while True:
    frames += 1
    _, frame = video_capture.read()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    canvas = detect(gray, frame)
```

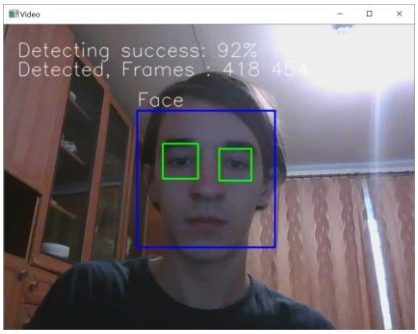
Also, we need to have a percentage value to see how good our program is working.

```
detSuccess = int(round(detected / frames,2) *100)
```

After that we need to see the results in our live window.

```
cv2.putText(frame,'Detecting success: ' + str(detSuccess) + '%',(20,50), font, 1,(255,255,255),1,cv2.LINE_AA)
cv2.putText(frame,'Detected, Frames : ' + str(detected) + ' ' + str(frames),(20,80), font, 1,(255,255,255),1,cv2.LINE_AA)
```

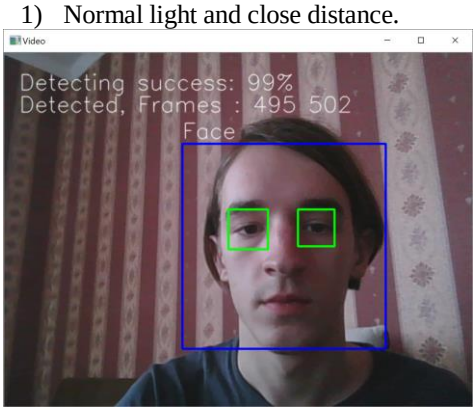
So now it will look like this.



Now I will test our program for quality in different situations.

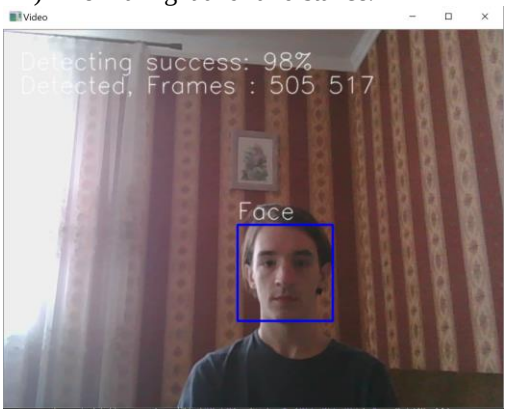
For the beginning I want to say that all the experiments will be made without 100% precision. These tests will show the average result of how this program works in different situation.

1) Normal light and close distance.



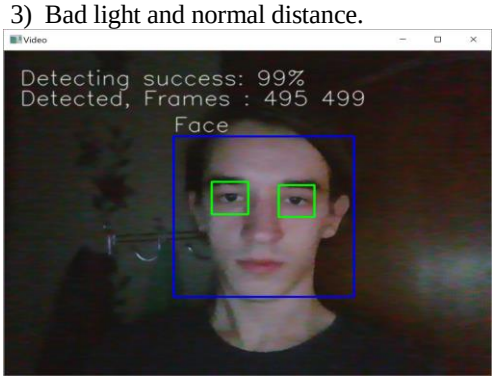
And as we can see, in this situation detecting success is very good, so in this situation our program works well.

2) Normal light and far distance.



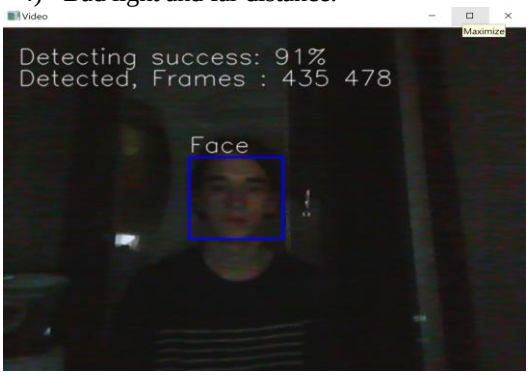
So, on this distance face detecting is still very good, but the eyes detecting does not work properly. Our program aimed at face detection, so this moment is not frustrating us very much. And we will continue.

3) Bad light and normal distance.



In this situation detecting success is also very good despite of bad light.

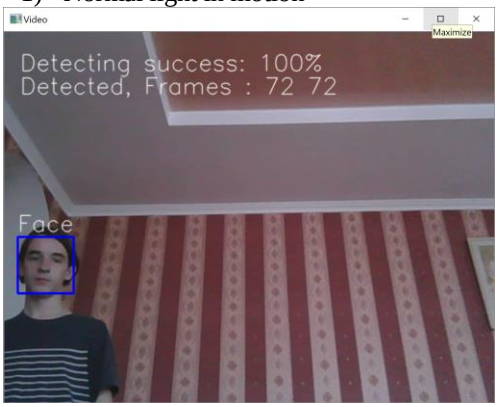
4) Bad light and far distance.



It is great surprise that our program works so good in such a bad condition. Detecting success is 91% despite of very bad light and a big distance. It is very good.

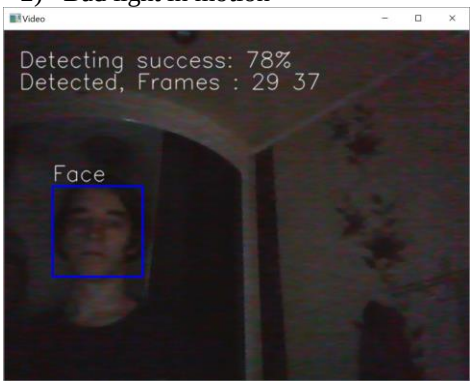
In the end, there will be 2 tests when I will move from the left side to the right in bad and good lights:

1) Normal light in motion



The result is great, 100% detecting success in motion. This test lasted near 5 seconds (others near 30 second, because this test needed moving from one side to the other), so I think that detecting success would be a little bit smaller, but still great. Eyes detection is still bad working on far distance.

2) Bad light in motion



So, the result is also good enough. And as you can see, in bad light our program works a little bit slower and returns less frames. This test is also lasted 5 seconds.

Also, I created table 1 with the results of our tests.

Table 1. The results of tests

	Normal light	Bad light
IN STATIC		
Close distance	99%	99%
Far distance	98%	91%
IN MOVING	100%	78%

6 Conclusions

The paper represents the algorithms and auxiliary methods used in Face Recognition using OpenCV library, program that performs live face recognition and tests of this program.

The scientific novelty of obtained results is that a lot of different companies can use Face recognition to check attendance of students, schoolchildren, workers automatically, face recognition can help us with new automatically door locks and a lot of other things. Computer vision has much a lot of ways to be used in our life.

The practical significance of obtained results is that the face recognition can help people with different problems and make our life easier.

Prospects for further research are to identify people faces and making the program more effective

References

[1] Zhang, C., Murayama, Y.: Testing local spatial autocorrelation using k-order neighbours. In: International Journal of Geographical Information Science, Vol-14, 681-692. (2000)

[2] Estivill-Castro, V., Lee, I.: Amoeba: Hierarchical clustering based on spatial proximity using Delaunay diagram. In: 9th Intern. Symp. on spatial data handling, 26–41. (2000).

[3] Kang, H.-Y., Lim, B.-J., Li, K.-J.: P2P Spatial query processing by Delaunay triangulation. In: International Workshop on Web and Wireless Geographical Information Systems, Vol-3428, 136–150. (2005).

[4] Boehm, C., Kailing, K., Kriegel, H., Kroeger, P.: Density connected clus-tering with local subspace preferences. In: ICDM '04 Proceedings of the Fourth IEEE International Conference on Data Mining, 27–34. (2004).

[5] Boyko, N., Shakhovska, N., Basystiuk, O.: Performance evaluation and comparison of software for face recognition, based on dlib and opencv library. In: Second International Conference on Data Stream Mining and Processing, 478-482. (2018).

[6] Tung, A.K, Hou, J., Han, J.: Spatial clustering in the presence of obstacles. In: The 17th Intern. conf. on data engineering (ICDE'01), 359–367. (2001).

[7] Boyko, N.: A look trough methods of intellectual data analysis and their applying in informational systems. In: International Conference on Computer Science and Information Technologies, CSIT, 183-185. (2016).

[8] Schroff, F.: FaceNet: A Unified Embedding for Face Recognition and Clustering". At: https://www.cv-foundation.org/openaccess/content_cvpr_2015/app/1A_089.pdf (2018)

[9] Face Detection Algorithms and Techniques. At: https://facedetection.com/algorithms (2018)

[10] Ramiz, R.: Face Detection Using OpenCV and Python. At: https://www.superdata science.com/opencv-face-detection/ (2017)