

# Performance Evaluation and Comparison of Software for Face Recognition, based on Dlib and Opencv Library

Nataliya Boyko

*Department of Artificial Intelligence  
Lviv Polytechnic National University  
Lviv, Ukraine  
Nataliya.i.boyko@lpnu.ua*

Oleg Basystiuk

*department of information systems and  
networks  
Lviv Polytechnic National University  
Lviv, Ukraine  
obasystiuk@gmail.com*

Nataliya Shakhovska

*Department of Artificial Intelligence  
Lviv Polytechnic National University  
Lviv, Ukraine  
Nataliya.b.shakhovska@lpnu.ua*

**Abstract**— Overview and investigate time complexity of computer vision algorithms for face recognition. Main article idea is to compare two popular computer vision libraries, they are OpenCV and dlib, explore features, analyze pros and cons each of them and understand in what situation each of them suit the best. Method. The technologies of computer vision, which are used for face recognition was worked out. Research of two popular computer vision libraries was conducted. Their features are analyzed and the advantages and disadvantages of each of them are estimated. Examples of building recognition application based on histogram-oriented gradients for face finding, face landmark estimation for face orientation, and deep convolutional neural network to compare with known faces. The article generalizes the concept of face recognition. The scientific basis for facial recognition and the construction of a complete recognition system was described. The basic principles of the programs for face recognition are formulated. A comparative analysis of the productivity of both libraries in relation to - the time of execution to the number of iterations of the applied algorithms was presented. Also built two simple applications for face recognition based on these libraries and comparing their performance.

**Keywords**— *computer vision, face recognition, algorithms, performance, dlib, openCV, machine learning, HOG, face landmark estimation, DCNN, SVM*

## I. INTRODUCTION

Nowadays technologies of artificial intelligence are actively developing; they open up huge possibilities in front of us. Analysis, forecasting, recognition came to a new level with the use of artificial intelligence and machine learning technologies. In recent years, an extremely promising field of research is computer vision. Technology in this area is most in demand in our everyday life.

One of the most popular computer vision problems that is being actively explored is face recognition. Moreover, the latest developments of technological leaders in the world are proof of this. The Apple Special Event in September presented the technology Face ID. The active introduction of recognition technology by social networks for tagging people in photos and Facebook as a leader in this direction. The algorithm developed by Facebook shows the recognition efficiency of 93%.

The result of this development was the creation of various types of libraries and APIs for face recognition.

Developed solutions especially for a particular area, or capable of solving several at once, written in a particular programming language or with the support of all popular languages. Often, among all the variety of tools, it's hard to understand which one is best suited for solving your problem.

This article will discuss computer vision, namely face detection based on dlib and OpenCV libraries. All the advantages and disadvantages of each of them will be appreciated, and the feasibility of using them in projects for constructing recognition systems. However, before the development and comparison of libraries, we need to find out the basic things and terms associated with this doctrine, we will also understand what tasks we face, as the developers of a successful software product for the face to face.

The research objective is to compare the performance of two popular computer vision libraries and build on their basis two simple recognition systems. To achieve this, the following tasks must be solved:

- consider the methods and basic principles of face recognition;
- analyze existing recognition technologies;

Perform a comparative analysis of the performance of OpenCV and dlib libraries based on the HOG method for searching and subsequent recognition to appear in Python.

## II. LITERATURE REVIEW

In the present, technologies of computer vision are actively developing, with their help, we can solve problems more effectively, one of which is recognition. As a result of active development, developers receive a large number of libraries to solve problems associated with computer vision. Actually, in this article we face the task of determining the performance of these libraries. The best way to get detailed information about a particular system, library, or API is to get acquainted with the documentation that we will actually use for the research in this article [1, 2]. Works [6, 7] focus on the theoretical aspects of building a stable system for face recognition.

The researchers [3, 5] describe the actual methods and technologies for all stages of the development of the recognition system, since in the field of recognition, a huge number of unique solutions have been developed. In any system, there is a promising issue of its performance, and

especially this applies to recognition systems, so the authors [4] investigate the speed of the operation of recognition methods. Also, scholars [9, 10] describe the method of recognition using the method of support vectors machine (SVM), which can significantly improve the speed of the recognition process. Researchers [8] describe face landmarks estimation algorithms that are used to position faces. It enables you to increase the quality of the system by aligning the face for better recognition.

Based on the conducted analysis it was established that for the creation of the recognition system there are no uniform methods and technologies that would combine all stages of construction. Therefore, the work with a detailed description of technologies and tools for their implementation, the assessment of the advantages and disadvantages of each of them and performance research to address such problems is relevant.

### III. MATERIALS AND METHODS

Before the developers of the recognition systems face, there are the following problems:

- Find faces - no matter whether the task of recognizing people in photographs, or video recognition, or anything else.
- Face positioning - photos are not often found on which a person stands directly in front of the lens, most often the face is turned, we face the task of positioning it as if the photo was taken directly.
- Defining unique facial features - this step can be called a full face recognition step (previous ones were preparatory), it analyzes the image and obtains unique digital values of the face.
- Identification of a person - we compare the received data with the data already available to us, if the data are similar, we will display the name of the person, if not, accordingly we have not known yet to us person.

This article will examine in detail all the steps to build a face recognition system and compare their implementation with the help of various libraries, as well as the speed of the work of each stage in different libraries of computer vision. Keep in mind that the research results may vary, depending on how the computer is filled with recognition. Also, all libraries that are used for research in this article are configured in the mode of use only CPU, without a GPU (NVIDIA CUDA).

The first step in the development of the system is important, since depending on how good the search was made. If we skip any face in this step, or we regard another object as a face in this step, then the recognition results can be considered unsatisfactory. One of the popular algorithms for finding facial expressions in the image was invented in 2005 by Navneet Dalal and Bill Triggs, HOG [2]. The algorithm's work consists of the following steps:

Calculation of gradient value - The most common way to calculate is to use a one-dimensional centered mask with subsequent filtering of the color or saturation of an image using such filtering kernels:

There are four methods of normalization, the best way to recommend this method:

$$\text{L2-norm: } f = \frac{V}{\|V\|_2 + e^2};$$

In formula: V is not normalized block vector,  $\|V\|_2$  - the norm of the vector, k - some small constant (the exact meaning is not important to us).

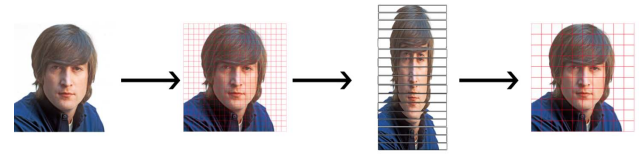


Fig. 1. Simplified visualization of the algorithm's work HOG.

After finding the face in the image, we face the task of positioning it, since in the vast majority of images, the faces are not centered, but are turned either from the angle and this will worsen any recognition later on. To solve this problem, use the face landmark estimation algorithm (face orientation estimation). The basic idea is to find 68 landmarks that are present on each face - the upper part of the chin, the inner edge of the eyebrows, the outer edge of the eye, the lower point of the nose, and the upper and lower points of the lip, and so on. After finding these landmarks with simple manipulations (rotation, increase or decrease), we get positioned in the center of the face.



Fig. 2. All 68 points of reference, which will recognize the algorithm on the face.

For positioning we will use the OpenFace additional library, it will perfectly "be friends" with the libraries we will use, it will look identical for both libraries, the performance will also be identical, so we will not compare it.

Here we come to the step that is directly related to the problem of face recognition. To distinguish between faces, we need to find unique features. And the first question that we face is how to organize this recognition. The easiest way to recognize faces is to compare the unknown face that was obtained in the second step with all the faces available in our database, hoping to find the same. Accordingly, during each recognition we refer to the database. The idea at first glance is good, but if we have huge data volumes, this will increase the sample time. Further consideration should be given to the option of passing these data to the cloud if we are talking about a large amount of data, which gives rise to the problem of delays between requests to and replies from the server. In addition, for most recognition projects, the speed of

recognition is in the first place, as you already understand, for such projects this method does not fit.

Because of the ineffectiveness and high computational complexity of the previous method, it was proposed to select several basic features of the face. We faced with the problem of choosing this rice. At first glance,, it seems that size or eye color, length of nose, size of the lips, eyebrows form - are the main characteristics by which a person recognizes faces. However, studies have shown that these features have no value to the computer face recognition. This problem is due to the fact, that the computer can not evaluate the face as a whole, but pixels the image.

To solve this non-trivial problem, it was suggested to use DCNN, which will be trained to identify 128 unique numerical facial features.

The process of training such a neural network works on the following principle:

- Upload the face image of a person we already know (classified).
- Upload another face image of the same person.
- Upload an image of another person's face.

The neural network will adjust the results of the obtained values so that 128 characteristics of the images loaded in steps 1 and 2 are as close as possible and the image loaded in step 3 differs from them as much as possible.

The idea of reducing the complex raw data to the list of computer generated numbers received the most development in the machine learning and was first used in the field of translation. The face-to-face approach, as suggested in the article, was firstly introduced by Google engineers in 2015.

The process of training the deep convolutional neural network is to generate unique numerical features of the face is a complex process that requires a huge database of faces and a significant computing ability of the computer.

Even using NVidia graphics card Telsa, since greater speed model train on NVidia graphics cards companies with presence support CUDA, the learning process took about 24 hours. However, as soon as the neural network is trained, it will be able to generate unique characteristics for any person, even for one that it has never seen before. So, this step is extremely important. The neural network is trained only once, but it will depend on the effectiveness of our face recognition system. If you can not train your own neural network to generate features, you can use an already trained network.[3].

The last step of the algorithm is to compare existing in our data, namely 128 features some faces that we received in the previous step, with all of our data about the people, if the data coincide, we can tell who is shown in the picture. This step can also be implemented in two ways:

- Database - as above described, with large volumes of data, this greatly reduces application performance. However, as before, this method of organization can be used for small volumes of data when it is a task of identifying a small group of people and there is not

enough photos for quality training of the method of reference vectors.

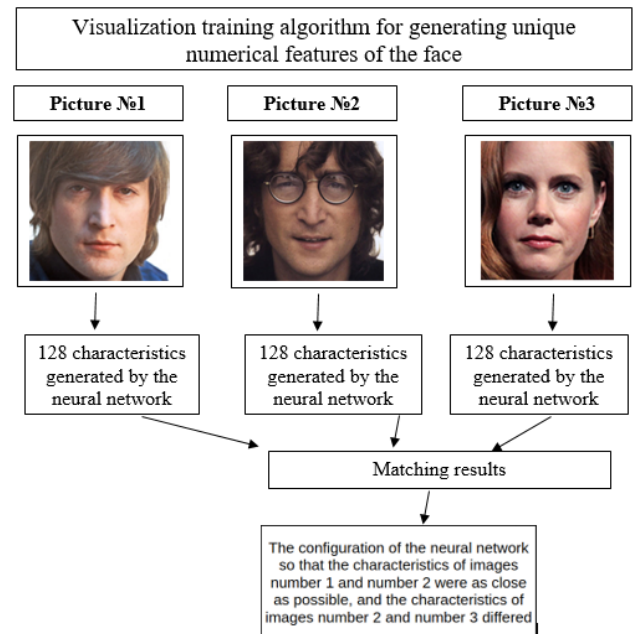


Fig. 3. Visualization of the learning algorithm to determine the unique numerical features of the face in the future.

- Support vector machine (SVM) - using existing features we train a classifier, the more homogeneous data set, the better for us because with this we can more accurately determine the face. Why the more homogeneous data, the better? For the classifier, we pass the value in the form of data set:

$$(\vec{x}_1, y_1), \dots, (\vec{x}_n, y_n),$$

where  $y$  is 1 or -1, and each one indicates the class to  $\vec{x}_i$  which the point  $\vec{x}_i$  belongs. During the training, support vector method, our challenge is to find the maximum separation hyperplane (see Fig.3), simply put, the dataset is divided into classes so that they are difficult to confuse during comparison. The classifier's working time is several milliseconds, ideal for face-to-face identification.

After weighing the pros and cons we will discuss identification method using support vector method. At this stage, there is also no reason to compare performance because the data set for learning is the same, accordingly, we create a classifier for two libraries, so at this point the speed will be the same for the two libraries.

Also, please note some features that can help you to improve the learning [7]:

- 1) Between the data (in our case, it's 128 facial features) there should be clear intervals so that they can be divided into groups, this depends on how well-trained the algorithm defines unique facial features.
- 2) Huge volumes of data will not provide the desired gain, but will only increase the learning time of the algorithm, to increase productivity, use data filtered

from the noise and any data from one class does not overlap the data of another.

#### IV. EXPERIMENTS

Let's make a study of all steps for building a recognition system based on the methods and technologies described above. Our first step is to search for the face based on the method proposed above, namely HOG. The result will be an array of coordinates for each found face in the photo.

An example of organizing this algorithm with dlib library tools:

##### Face\_detector\_dlib.py:

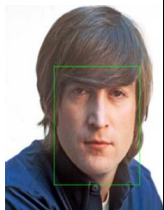

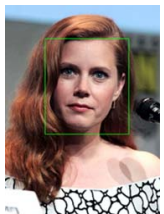
```
# Import the system and dlib libraries
import sys
import dlib
from skimage import io
# Call the basic function library to find faces
face_detector = dlib.get_frontal_face_detector()
# Go through all entered images
for images in sys.argv[1:]:
    print("Processing image: {}".format(images))
    image = io.imread(images)
    faces_detected_image = face_detector(image, 1)
    io.imshow('face_recognition.jpg', faces_detected_image)
    print("Number of faces detected: {}".format(len(
faces_detected_image)))
# Print the result
win = dlib.image_window()
win.clear_overlay()
win.set_image(image)
win.add_overlay(dets)
#Wait for "Enter" button to continue
dlib.hit_enter_to_continue()
```

An example of organizing this algorithm with OpenCV library tools:

##### Face\_detector\_opencv.py:

```
# Import the system and OpenCV libraries
import sys
import cv2
from skimage import io
# Call the basic function library to find faces
lbp_face_cascade = cv2.CascadeClassifier('data/
lbpcascade_frontalface.xml')
# Go through all entered images
for images in sys.argv[1:]:
    print("Processing image: {}".format(images))
    image = io.imread(images)
    faces_detected_img = detect_faces(lbp_face_cascade, image,
scaleFactor=1.2)
    io.imshow('face_recognition.jpg', image)
    print("Number of faces detected: {}".format(len(faces))
# Print the result
plt.imshow(convertToRGB(faces_detected_img))
#Wait for "Enter" button to continue
cv2.waitKey(0)
```

The results of the performance are as follows:

	Experiment №1	Experiment №2	Experiment №3
OpenCV			

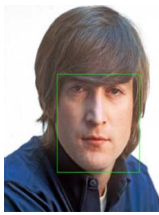

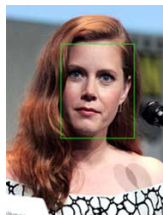
Recogniti on time (sec.)	0.5652	0.0934	0.6121
dlib			
Recogniti on time (sec.)	1.6584	0.2163	1.6889

Fig. 4. Comparison of the results and the time spent searching for faces using OpenCV libraries and dlib

As we can see at this step, the OpenCV library demonstrates better performance.

Going to the next step where we position the found face, I remind that for this we will use the face landmark estimation method, which we build on the basis of an additional OpenFace library.

##### Face\_landmark\_estimation.py:

```
# Import the system and OpenFace libraries
import sys
import dlib
import openface
from skimage import io
# Call the basic function library to find faces
face_detector = dlib.get_frontal_face_detector()
face_pose_predictor = dlib.shape_predictor(predictor_model)
face_aligner = openface.AlignDlib(predictor_model)
# Go through all entered images
for i, face_rectangle in enumerate(detected_faces):
    print("- Face #{} found at Left: {} Top: {} Right: {} Bottom: {}".format(i,
face_rectangle.left(), face_rectangle.top(), face_rectangle.right(),
face_rectangle.bottom()))
    pose_landmarks = face_pose_predictor(image, face_rect)
    alignedFace = face_aligner.align(534, image, face_rect, landmarkIndices
= openface.AlignDlib.OUTER_EYES_AND_NOSE)
# Save the result
io.imshow('aligned_face_{}.jpg', image, alignedFace)
```

The results are as follows:



Fig. 5. Visualization of the algorithm positioning faces

After receiving the characteristics, we only find such characteristics from the available data.

As described above, we will use SVM for this. Here's what an advertising implementation looks like:

##### SVM\_example.py:

```
import sys
import numpy as np
from sklearn import svm
x = np.vectors();
y = np.array();
x.append(np.vector([1, 2, 3, -1, -2, -3]))
y.append(+1)
x.append(np.vector([-1, -2, -3, 1, 2, 3]))
y.append(-1)
svm = dlib.svm_c_trainer_linear()
```



```

svm.be_verbose()
svm.set_c(10)
classifier = svm.train(x, y)
print("prediction for first sample: {}".format(classifier(x[0])))
print("prediction for second sample: {}".format(classifier(x[1])))
with open('saved_model.pickle', 'wb') as handle:
    pickle.dump(classifier, handle)

```

## V. RESULTS

The final version of the experiment are two face detection systems, one is based on a dlib library, the second is based on OpenCV. The results of their work submitted to fig.6, which provides an interface of applications, in which we see the detected face.



Fig. 6. Example of Face Detection

Based on the analysis of the submitted OpenCV and dlib libraries, it has been established that there is no single methods and technologies to create a distributed recognition information system that would combine all stages of the system construction. There is a huge number of methods for searching, positioning and in general for organizing the recognition process. By choosing technologies you should be very careful because depending on your needs you should use certain methods. Therefore, work with a detailed description of technologies for recognition is extremely relevant at present, as well as the development of new technologies and a way to solve this pressing problem.

## VI. DISCUSSION

As mentioned above, currently the technology of computer vision and tools for face recognition are actively developing. For qualitative recognition, faces are developed not only software products but also unique hardware solutions. Global technology leaders invest in research and development a lot of money because this problem has wide opportunities for use. The direction for the development of face recognition is primarily the protection and increased protection systems. Currently the

most used in entertainment, I believe this trend will grow also. Proposed approach can be used for big data processing [11].

## VII. CONCLUSIONS

As has been researched in this article, the OpenCV library is more productive, has better performance for face detection and detection. It also means that with OpenCV, it's better to build recognition applications for the IoT platform. Note that only HOG algorithm has been explored while searching for other algorithms, such as the Haar cascade, it works longer, but works out more in detail, if there are plenty of photos in the future for many, it is advisable to think about using this method. However, the design logic and the key points of creating an application recognition, was discussed in this article.

## REFERENCES

- [1] OpenCV: OpenCV Tutorials [Electronic resource] – Access mode: [https://docs.opencv.org/master/d9/df8/tutorial\\_root.html](https://docs.opencv.org/master/d9/df8/tutorial_root.html)
- [2] Dlib Python API Tutorials [Electronic resource] – Access mode: <http://dlib.net/python/index.html>
- [3] Z. Rybchak, and O. Basystiuk, “Analysis of computer vision and image analysis technics,” ECONTechMOD: an international quarterly journal on economics of technology and modelling processes, Lublin: Polish Academy of Sciences, vol. 6, no. 2, pp. 79-84, 2017.
- [4] Face Detection Algorithms and Techniques [Electronic resource] – Access mode: <https://facedetection.com/algorithms/>
- [5] A toolkit for making real world machine learning and data analysis applications [Electronic resource] – Access mode: [https://github.com/davisking/dlib/blob/master/python\\_examples/face\\_detector.py](https://github.com/davisking/dlib/blob/master/python_examples/face_detector.py)
- [6] R. Raja, Face Detection Using OpenCV and Python [Electronic resource] – Access mode: <https://www.superdatascience.com/opencv-face-detection/>
- [7] R. Raja, Face Recognition Using OpenCV and Python [Electronic resource] – Access mode: <https://www.superdatascience.com/opencv-face-recognition/>
- [8] A. Rosebrock, Facial landmarks with dlib, OpenCV, and Python [Electronic resource] – Access mode: <https://www.pyimagesearch.com/2017/04/03/facial-landmarks-dlib-opencv-python/>
- [9] Support Vector Machines [Electronic resource] – Access mode: <http://scikit-learn.org/stable/modules/svm.html>
- [10] F. Schroff, D. Kalenichenko, and J. Philbin, FaceNet: A Unified Embedding for Face Recognition and Clustering [Electronic resource] – Access mode: [https://www.cv-foundation.org/openaccess/content\\_cvpr\\_2015/app/1A\\_089.pdf](https://www.cv-foundation.org/openaccess/content_cvpr_2015/app/1A_089.pdf)
- [11] N. Shakhovska, “The method of Big data processing,” XII International Conference on Computer sciences and information technologies (CSIT), Lviv, Ukraine, pp. 122-125, 2017