Face Detection Using an Implementation Running in a Web Browser

Halyna Klym
Specialized Computer System Dpt.,
Lviv Polytechnic National University,
Lviv, Ukraine
klymha@yahoo.com, halyna.i.klym@lpnu.ua

Ivanna Vasylchyshyn
Theoretical and General Electrical Engineering Dpt.,
Lviv Polytechnic National University,
Lviv, Ukraine
ivanna.i.vasylchyshyn@lpnu.ua

***Abstract*—A face recognition system using machine learning has been developed. Node.js, Face-recognition.js, Dlib technology stacks were chosen to implement the system. The choice of means of realization of the client and server parts is carried out, the architecture of the application is defined. A desktop application has been created. An interface based on the HTML framework of the Twitter Bootstrap framework was developed and a comprehensive system testing was performed.**

***Keywords*—face recognition system, machine learning, Node.js library**

I. Introduction

Today one of the most promising areas in the field of information technology is machine learning and artificial intelligence [1-3]. These technologies have proven themselves at their best in classification problems, one of which is the recognition of human faces.

Making a computer see is not a new task and naturally fits into the concept of artificial intelligence. The main thing here is algorithms for image processing, both static and moving.

Face recognition systems are used in various fields [4-6]. However, there are still two main problems. Firstly, any image is an array of pixels. This makes this representation of the pictures excessive and uneconomical. Therefore, for effective recognition faces need to develop a certain compact and convenient format presentation of pictures. To date, many ways are known lossy image compression, but the format used is inconvenient to classify photos of people. The second problem is that the same person can be photographed under various external factors. In this regard, many image recognition algorithms have been developed, including the method of mapping graphs [7], the principal components method [8,9], the active appearance model method [10,11], etc.

However, any system requires ease of use. The system developed in this work provides an opportunity to recognize faces and identify a person in the image, which can be used as an additional level of authorization.

The main option of proposed system is to recognize and build a structural model of the face. On the basis of the received data identification from the set existing in a database is carried out. Due to a number of problems, the Node.js library has been modified for maximum face recognition.

II. Choice of Technologies for System Implementation

Node.js, Face-recognition.js, Dlib technology stack was chosen to implement the system. The Dlib library (unlike OpenCV) allows us to recognize not only the face in full face, but also with the angle of rotation from the camera. This demonstrates the high speed and quality of recognition.

Node.JS is a completely stand-alone platform that includes (in addition to the V8 engine) a built-in server (HTTP and TCP / UDP / Unix-socket), a basic set of libraries, and provides completely asynchronous work with files and network devices.

The main feature of Node (apart from complete asynchrony) is its single-threaded model. In other words, all operations are performed in the same OS thread, even if the server has a thousand simultaneous users. To implement multithreading and use all the cores of modern processors, it is recommended to simply download a few copies of the program. But adopt WebWorker from the HTML 5 standard and distribute the program to several child processes should be performed. The web application does useful work very quickly, and most of the time it just waits (data from the database, new-fangled NoSQL database), or just keeps open connections for Comet in memory. Therefore, thousands of requests can be processed in one thread without resorting to clustering. In Fig. 1 a comparative analysis with such well-known languages for server programming as Java and Ruby is shown. The results indicate that the selected Node.js is able to quickly return a response from the server, such as 200/OK.

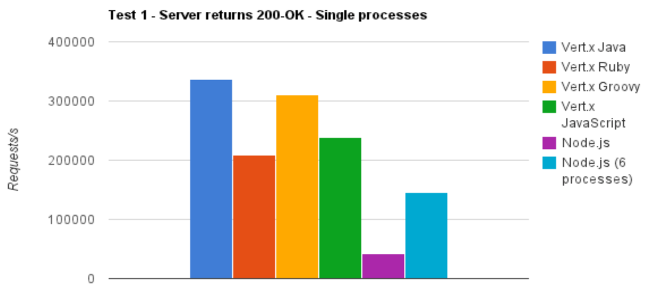


Fig. 1. Comparative analysis of technologies.

The second feature of the Node architecture is the fullness. Almost every operation has callbacks, generates events, and the user has an EventEmiter object, through which you can literally generate your events in one line (this is easy, because the event is just a line with the name, as well as a list of parameters passed to the handler).

The Dlib library is less well known than OpenCV, but has a number of features on a par with OpenSource. The library includes several implementations of SVM, implementation of neural networks, a selection of functions of matrix geometry, a selection of gradient descent algorithms, SURF | HOG.

Face-recognition.js - npm package, which allows you to quickly start recognizing faces with a simple API, if necessary, allows you to make detailed settings, easy to install.

III. General Steps for System Implementation

So, the proposed npm package uses the Dlib library, which provides Node.js binding for recognition tools within this library. Dlib uses deep learning techniques and comes with already trained models that have demonstrated recognition accuracy of 99.38% when using the LFW benchmark.

The steps of the face recognition are the following. Step 1: data preparation. To do this, 20 people of each character in different angles were gathered. 10 faces were taken for training, and the rest were used to assess the accuracy of recognition.

Step 2: defining faces. Face images of 150 x 150 pixels were pre-cut using opencv4nodejs. Also, faces can be cut, saved, marked with Face Recognizer.

Step 3: learning to recognize. The selected faces are transmitted to a neural network, which issues a descriptor for each face and stores it in the appropriate class. By specifying numJitters as the third argument, rotation, scaling, and mirroring can be applied. In this case, different versions of each of the input faces are created. Increasing the number of modifications can increase the accuracy of recognition, but the neural network takes a little longer to learn. The state of the recognizer is saved so that we do not download it again, but download it from a file. Recognizer is a deep learning neural network that uses unique facial descriptors to calculate. We can train the recognizer on an array of tagged face images, after which will be able to mark faces on the input image.

Step 4: recognition of new faces. Now, with the help of control data, the recognition accuracy is checked and the results are saved. Recognition is performed as follows. First, the Euclidean distance of the descriptor vector to each class descriptor is calculated for the input face, and then the average value of all distances is calculated. In certain cases, for this task, clustering by k-means or SVM-classifier is better. However, in this case, the speed and efficiency of the Euclidean distance is sufficient. When calling predictBest the result with the smallest Euclidean distance, i.e. with the greatest similarity is obtained. Even a small sample of training data allows us to get a fairly accurate recognition. And this despite the fact that some of the input images may be blurred due to the small size.

IV. Web Application Architecture

Visual Studio Code was chosen to develop the Web application. The overall architecture of web applications has a number of disadvantages, including poor scalability (supports only a single node.js process, cannot be distributed according to the number of rooms created, and cannot function separately without the server logic of the application); redundant code (there is a simple encapsulation of Socket.io, only on the server side the general architecture requires 400 lines of code as opposed to 100 lines with a distributed architecture. These shortcomings can be overcome using the distributed architecture in this work.

The distributed architecture of the developed Web application is shown in Fig. 2. In this architecture, the front-end server is responsible for connecting to the client, the chat server is responsible for processing the server logic of the application. This scalable architecture has significant advantages, namely: load sharing (the architecture completely separates the socket connection code from the server logic code. This is required in resource-intensive applications with a large number of asynchronous connections, such as anonymous chat); simplifies channel switching (users can change the caller without reconnecting to WebSocket through this separate architecture); scaling (several connect processes can be started and use hashing to display on the server).

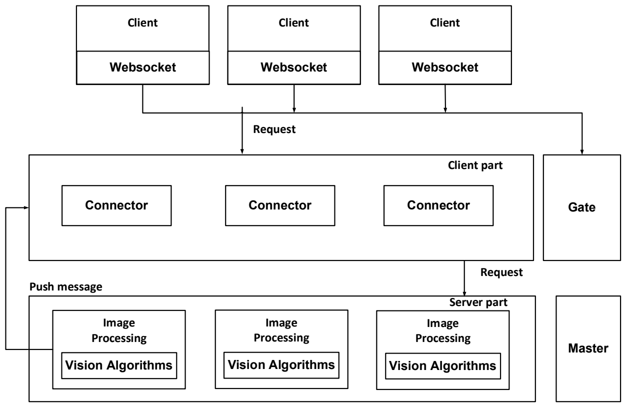


Fig. 2. Software solution architecture.

The implementation of this architecture has the advantage of fewer lines, which takes up server code compared to the usual Node.js architecture of applications that use only one Node.js process processing all requests from WebSocket.

V. Face Recognition System

The face recognition system is implemented on the basis of face-api.js. It is a JavaScript module built on the tensorflow.js kernel that implements several CNNs (Convolutional Neural Networks) for face detection, recognition, and face landmark detection. The module is optimized for the Internet and for mobile devices.

Traditionally, faces-recognition.js is used, but today there is no library for reliable face detection and recognition, which can be fully launched in the browser. Therefore, in this paper, the library is implemented for its intended purposes using tfjs-core. This allows us to get almost the same results as face-recognition.js, but in a browser. Face-api.js offers models optimized for working on the Internet and running on mobile resources. There is no need to install external components. It is also possible to turn on the GPU, start operations on the backend WebGL.

A. *Implementation of Face Recognition Using Machine Learning*

The main task of the system is to identify the person who received the image of face, such as the input image. This section describes the approach used in face-api.js to implement face recognition.

It is first necessary to provide one (or more) images for each person we want to recognize. After that, the person's name must be marked. The input face is now compared to the reference data to find the most similar images. If both images are sufficiently similar, the person's name is displayed, otherwise "unknown" is displayed.

At implementing the system, two problems need to be solved. First, an image can contain several faces, so all of them need to be detected. Second, the metrics of similarity of the two faces for their comparison must be obtained. To solve the first problem, all the faces in the input image must be found using Face-api.js, which implements several face detectors for different versions. The most accurate face detector is the SSD (Single Shot Multibox Detector), which mainly consists of CNN based on MobileNet V. However, some additional layers of prediction of the box are stacked on top of the network. Face-api.js implements an optimized tiny face detector (basically a uniform version of Tiny Yolo v2 that uses deeply highlighted convolutions instead of regular convolutions). MTCNN (multitasking cascade converting neural network) should be applied, which in most cases is used for experimental purposes.

The algorithm of the system using the limiting boxes is shown in Fig. 3.

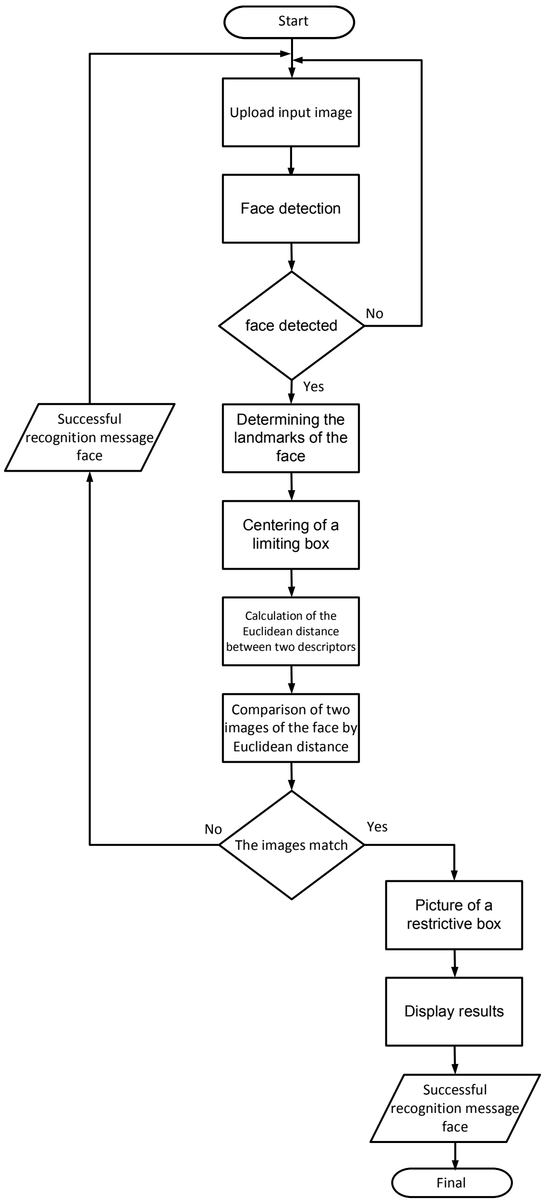

Start
Upload input image
Face detection
face detected
No
Yes
Determining the landmarks of the face
Centering of a limiting box
Calculation of the Euclidean distance between two descriptors
Comparison of two images of the face by Euclidean distance
The images match
No
Yes
Successful recognition message face
Picture of a restrictive box
Display results
Successful recognition message face
Final

Fig. 3. Algorithm of system operation.

Networks return the bounding boxes of each face with the appropriate scores, such as the probability of each bounding box showing the face. Estimates are used to filter the bounding boxes, as it may be that the image does not contain faces at all (Fig. 4).

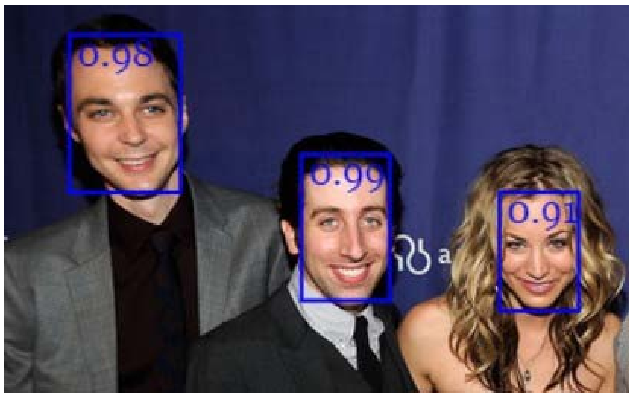


Fig. 4. The result of determining the faces by imposition of limited boxes.

B. *Defining and aligning face landmarks*

For more accurate face recognition, the border boxes need to be aligned so that the image can be extracted. To do this, face-api.js implements a simple CNN, which returns the landmarks of 68 points of the depicted face (Fig. 5). The bounding box of the face centers.

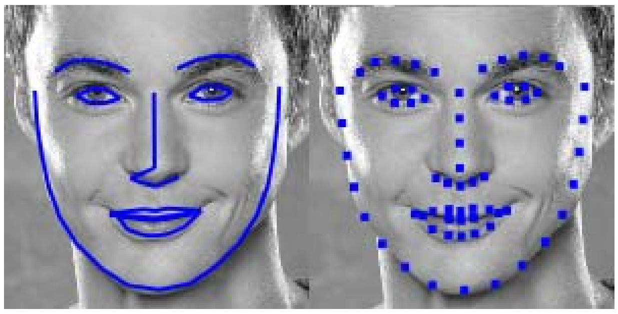


Fig. 5. Placement of landmarks for centering the limiting box.

In Fig. 6. the results of face detection (left) compared to the aligned face image (right) are shown.

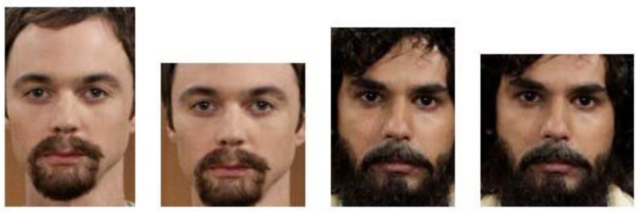
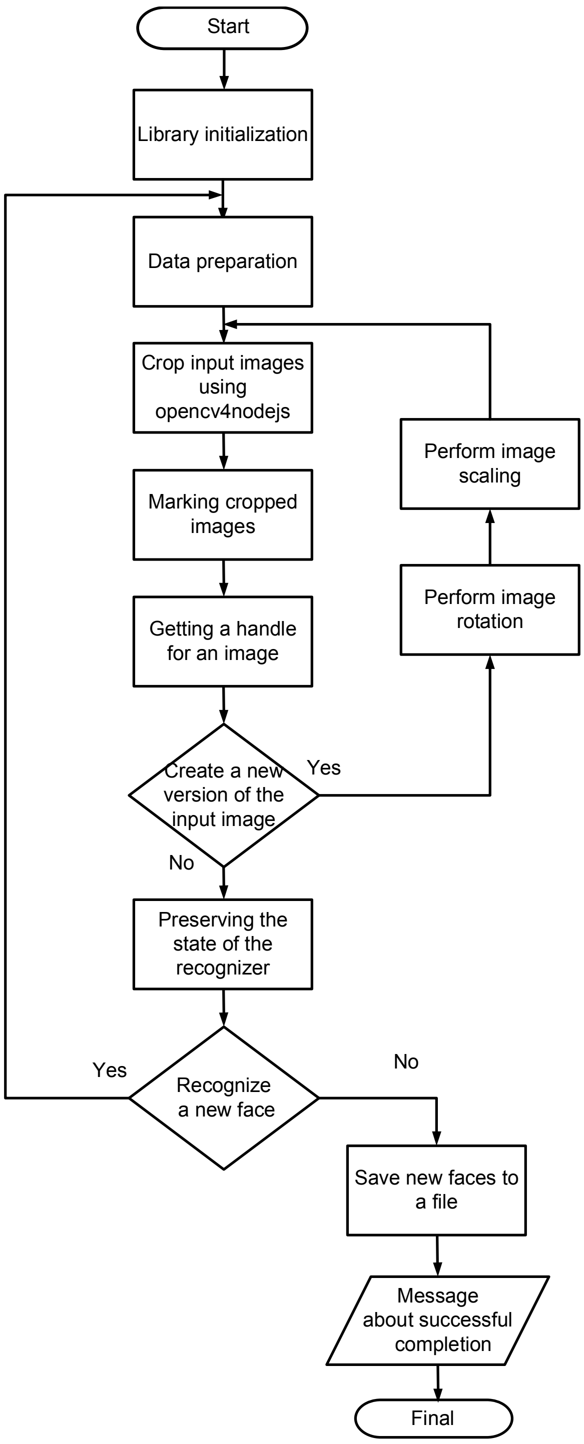


Fig. 6. Comparison of the detected face with the centered one.

The extracted and aligned face images were then transmitted to a recognition network based on a similar ResNet-34 architecture and corresponding to the architecture implemented in dlib. The network has been trained to calculate the characteristics of the human face to the face descriptor.

The algorithm for learning the recognizer is shown in Fig. 7.



Start
Library initialization
Data preparation
Crop input images using opencv4nodejs
Marking cropped images
Getting a handle for an image
Create a new version of the input image
Yes
No
Perform image rotation
Perform image scaling
Preserving the state of the recognizer
Recognize a new face
Yes
No
Save new faces to a file
Message about successful completion
Final

Fig. 7. Recognition algorithm.

To match two faces, the face descriptor of each deleted face image will be used. Comparisons will also be made with the face descriptors of the reference data. In other words, the Euclidean distance between two face descriptors will be determined based on a threshold value (for example, a threshold value of 0.6). Fig. 8. visualizes the comparison of two images of the face at Euclidean distance. To do this, in the implemented system in dist / face-api.js added <script src = "face-api.js"> </script>.

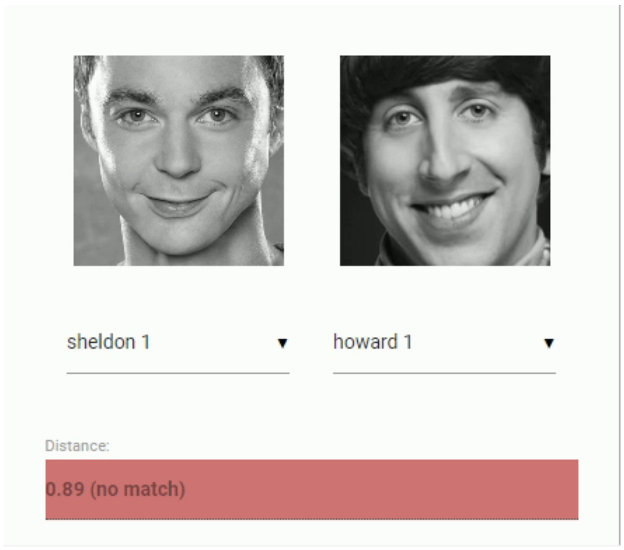


Fig. 8. The result of comparing of different faces.

VI. Conclusion

The system for face recognition using machine learning using the Node.js platform is implemented. It is possible to run in a browser, as well as on any VPS. At the same time the client-server architecture is used.

References

[1] D. W. Bates, A. Auerbach, P. Schulam, A. Wright, and S. Saria, “Reporting and implementing interventions involving machine learning and artificial intelligence,” Annals of Internal Medicine, vol. 172, pp. S137-S144, 2020.

[2] V. Dunjko, and H. J. Briegel, “Machine learning & artificial intelligence in the quantum domain: a review of recent progress,” Reports on Progress in Physics, vol. 81(7), pp. 074001, 2018.

[3] M. G. Kibria, K. Nguyen, G. P. Villardi, O. Zhao, K. Ishizu, and F. Kojima, “Big data analytics, machine learning, and artificial intelligence in next-generation wireless networks”, IEEE access, vol. 6, pp. 32328-32338, 2018.

[4] M. Sharif, F. Naz, M. Yasmin, M. A. Shahid, and A. Rehman, “Face Recognition: A Survey,” Journal of Engineering Science & Technology Review, vol. 10(2), 2017.

[5] M. He, J. Zhang, S. Shan, M. Kan, and X. Chen, “Deformable face net for pose invariant face recognition,” Pattern Recognition, vol. 100, pp. 107113, 2020.

[6] M. Jacquet, and C. Champod, “Automated face recognition in forensic science: Review and perspectives,” Forensic Science International, vol. 307, pp. 110124, 2020.

[7] M. Qi, W. Li, Z. Yang, Y. Wang, and J. Luo, “Attentive relational networks for mapping images to scene graphs,” Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 3957-3966, 2019.

[8] F. A. de Almeida, A. C. O. Santos, A. P. de Paiva, G. F. Gomes, and J. H. de Freitas Gomes, “Multivariate Taguchi loss function optimization based on principal components analysis and normal boundary intersection,” Engineering with Computers, pp. 1-17, 2020.

[9] M. Iqtait, F.S. Mohamad, and M. Mamat, “Feature extraction for face recognition via Active Shape Model (ASM) and Active Appearance Model (AAM),” IOP Conference Series: Materials Science and Engineering, vol. 332, pp. 012032, 2018.

[10] S. Lombardi, J. Saragih, T. Simon, and Y. Sheikh, “Deep appearance models for face rendering,” ACM Transactions on Graphics (TOG), vol. 37(4), pp. 1-13, 2018.

[11] Z. H. Feng, J. Kittler, B. Christmas, and X. J. Wu, “A unified tensor-based active appearance model,” ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM), vol. 15(3s), pp. 1-22, 2019.