IRSTI 28.23.29 https://doi.org/10.26577/1i32jpcsit2302

S.D. Aitzhanov, N.M. Kazyieva, N.A. Burambaeva, Z.B. Shuren*, A.A. Aikeyeva

L.N. Gumilyov Eurasian National University, Astana, Kazakhstan
*e-mail: adai_zhazi@mail.ru

APPLICATION OF FACENET MACHINE LEARNING MODEL AND HAAR CASCADE CLASSIFIER FOR BIOMETRIC IDENTIFICATION

Abstract. The paper analyzes the principles of Haar cascade classifier and FaceNET machine learning simulation program for biometric identification. As an experiment, a recognition system was created that will allow human face recognition, which was developed in the Python programming language. Some libraries covered in the research process include numpy, OpenCV, pip, matplotlib, virtualenv and pickle. The Haar cascade classifier is used to detect objects in images and videos. For the process of generating a facial signature, the two programs used the FaceNET machine learning model, which uses convolutional neural networks in the process of extracting features from facial images and conducting a comparative analysis of them between the identification data and those stored in the database. Due to this, FaceNET identifies faces in images with high accuracy and security, which is useful for use in access control systems, automation of visitor accounting processes and other applications where facial recognition is necessary. The created system will provide an opportunity to recognize unique facial features, store them and link them with the user's documentary information.

Key words: biometric identification, machine learning model, facial image, Haar cascade classifier, face signature.

1. Introduction

Biometric methods, while being an effective means of protecting information, have risks associated with their use, such as imperfect scanning technologies, the possibility of theft and tampering of biometric information or photos, and many others [1, 2].

To solve the security problem and improve the performance of biometric methods, more and more researchers are applying neural networks.

The proposed architecture is based on convolutional layers, as convolutional neural networks have high accuracy and robustness to various distortions when applied in the field of facial biometrics. As an example, a system is developed to perform human face recognition. The created program has a small code size in Python language. The developed system uses FaceNET machine learning model and Haar cascade classifier, which are based on deep neural networks that allow face detection and recognition. The application of convolutional neural networks is necessary to extract facial features from images. These features are high dimensional vectors that describe the unique characteristics of each face and allow for comparative analysis of two faces to determine if they belong to the same person.

The model utilizes a training method called "one-shot learning", which allows training with the smallest number of images, i.e., one image is sufficient. This model can be implemented using small computational resources. The Python language and some number of defined programs are used to implement this program. These include *numpy, OpenCV, Jupyper Notebook, pip, matplotlib, virtualenv*, and *pickle*. With a number of specific operations performed, *virtualenv* must be installed and run, as this program will allow you to create a virtual environment for OpenCV. Next, a virtual environment, named as OpenCV, was created and activated for use. Activation is performed by a bat-file located in the Scripts directory of the created OpenCV virtual environment. The *numpy* program was installed next, followed by the OpenCV library itself and the *Jupyter Notebook* program. The library provides image-processing capabilities, and the program is an interactive notepad that allows you to work with many runtime environments.

2. Applications of the Cascade Haar Classifier

The input of the FaceNET machine-learning model is faces of size 160*160 pixels. The FaceNET system is capable of outputting 128 values, which are individual indicators of features unique to that face. In the case of an available photograph of a face in a different perspective, these values may differ, but not significantly, giving the same selection of outputs with almost identical numbers. The resulting 128 values are a face «signature» that is unique to each face. The input photo has not only a face image, but may also contain other objects and faces, or will have no face at all. Therefore, a procedure is needed to detect faces in the images, which leads to the need for a cascaded Haar classifier. Given an image as input, the classifier allows to determine the presence of faces, the location of a face and the number of located faces. The Haar classifier performs calculations by applying the so-called Haar functions, which are calculations for contiguous rectangular regions at a particular location in the detection window. Also included in the calculation process is summing the pixel intensities in each region and calculating the distinguishing characteristics between the sums. Currently, over 6000 features are correlated with the image and based on the correlation between the image and the features. Figure 1 shows an example of a general representation of Haar classifier training.

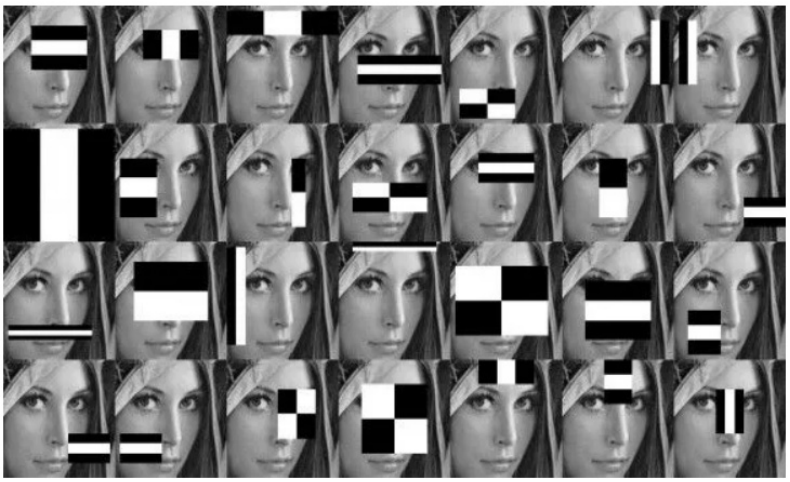

Figure 1 – A general representation of the training of the Haar classifier [3]

The two-step process results in the creation of two program codes for each step. The first program is designed only for the creation of a database of face signatures, while the second will allow the recognition of all faces. The first stage involves the collection of face images, followed by the creation of a database based on the production of corresponding "face signatures". The goal of the first program is to create a database consisting of signatures of the faces we want to recognize. Having only one photograph of each person is sufficient. By having more than one photo of each person available, the accuracy can be improved. The images are processed by a Haar cascade classifier, followed by FaceNET, which will produce «face signatures» that are stored in the database. The number of face signatures required directly depends on the number of face images of different people. The size of 160*160 pixels is used, as this size is the optimal size for model training. A database can be stored in different types of file formats and in this case, the database is saved in *picklefile* format, in particular, this database is saved as data.pkl. Figure 2 shows a schematic of signature generation based on the input stock image [4].

The second step involves the code generation process and consists of capturing an image from the workstation camera, directing the input image from the camera to the Haar cascade classifier, feeding the transformed image to the FaceNET model to generate a signature unique to that person. This signature of the face is compared with already existing records in the database. Figure 3 shows the scheme of face signature generation based on the input image from the camera.

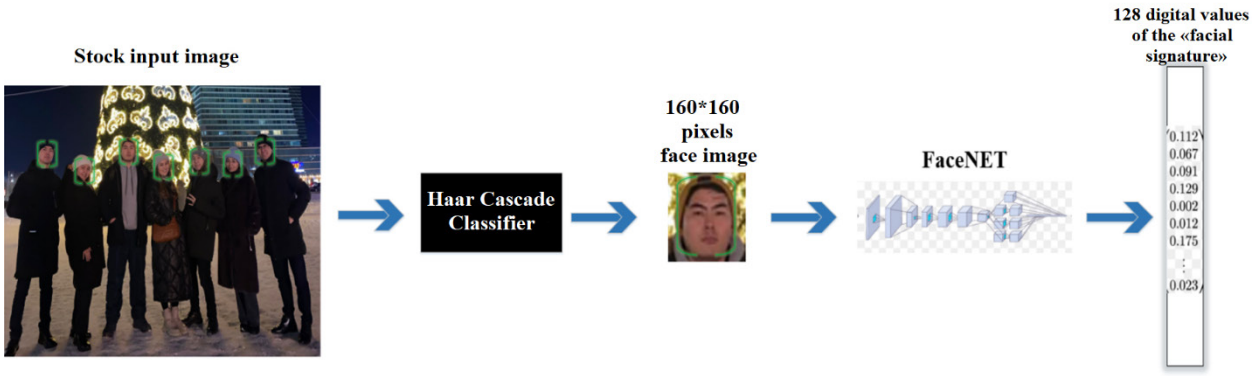

Figure 2 – Scheme of face signature formation on the basis of input stock image

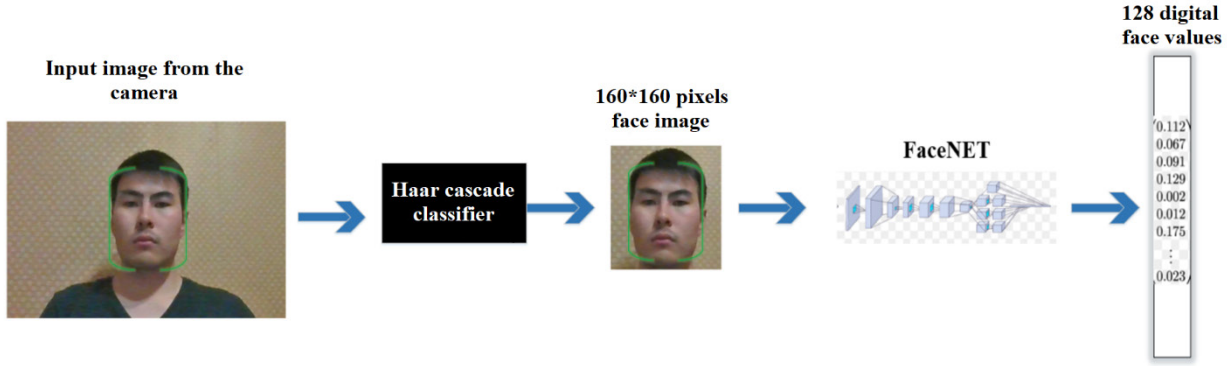

Figure 3 – Scheme of face signature generation based on the input image from the camera

Frobenius norm is applied to compare face signatures. In this case, each signature has 128 numerical values and can be represented as a vector with 128 numerical values. The *NumPy* library has many other functions for working with matrices and their norms. This library provides possibility to compute Frobenius norms for several matrices simultaneously. When applying the FaceNET model, a vector of 128 numerical values is formed, which provides a description of the unique characteristics of a given face in the image, and the values of the vector can have both positive and negative values, and their interpretation in most cases does not make physical sense. The process of generating 128 values when applying the FaceNET model consists of 3 steps:

- The face image is fed to the input of the FaceNET machine learning model;
- FaceNET machine learning model performs image processing and returns a vector of numerical values of length 128;
- The obtained vector is a «face signature» for the person whose image was fed to the input of the FaceNET machine-learning model.

Figure 4 shows a fragment of the generated 128 numerical values of the «face signature» [5].

The values in the database are labeled as «signatures» for simplicity, and the values obtained during the processing of the camera image are named as «value» and the norm value is N. The following method is applied for calculation as given in formula 1:

$$N = \frac{1}{128}\sum_{1}^{128}\sqrt{(signatures - values)^2}, \quad (1)$$

where N is the Frobenius norm.

This number is the norm that is applied to find which of the face «signatures» of the database is similar with the «value». And the calculations go accordingly with the arrangement of values, i.e. the first value of «signature» with the first value of «value». Based on these calculations it is possible to determine the smallest value of the norm, which is the correspondence for the signature from the database with the signature from the camera input image.

```
[9]: {'Aitzhanov Serik': array([[ 3.70360389e-02,  2.68896762e-02,  3.11047677e-02
                1.12144411e-01,  9.67933889e-03,  3.85443047e-02,
               -5.55540174e-02,  1.46011589e-02, -5.52674532e-02,
                1.53160309e-02,  1.00477464e-01, -4.25928235e-02,
               -1.45726511e-02,  3.42260525e-02,  6.08078986e-02,
                1.17224315e-02,  4.93489504e-02,  1.22434713e-01,
               -4.92296293e-02,  3.99966650e-02,  1.82312715e-03,
                1.10424384e-02,  2.90837418e-02, -6.63842726e-03,
               -1.25977378e-02, -4.19195518e-02,  5.46452999e-02,
                4.80781049e-02,  5.98546527e-02, -1.06499856e-02,
                1.04589369e-02,  2.79925037e-02, -1.33225685e-02,
                8.51024836e-02, -4.14528027e-02, -1.01784514e-02,
                4.92871832e-03,  4.47172020e-03, -1.41953183e-02,
               -4.13080864e-03,  3.73793859e-03, -1.40366387e-02,
                5.07293753e-02,  3.53181250e-02, -7.02813501e-03,
                2.93012592e-03,  5.11383265e-02, -2.84142420e-03,
               -1.02356058e-02,  3.98683585e-02,  3.11332885e-02,
                2.84698494e-02,  9.27359685e-02,  6.06311969e-02,
                4.90146205e-02, -1.29025457e-02, -1.57244839e-02,
                3.90704302e-03,  4.65833172e-02, -1.98914167e-02,
                2.67062280e-02, -2.62866751e-03, -3.90950739e-02,
               -3.27188410e-02,  6.45736307e-02,  1.17638921e-02,
               -3.28279585e-02, -1.06740482e-02,  6.80465326e-02,
                3.10044698e-02, -3.31488028e-02,  4.55110148e-02,
               -4.67289723e-02, -2.31712721e-02,  5.75695187e-02,
                2.78447848e-02, -3.34302485e-02, -2.79000700e-02,
               -5.89689054e-02, -9.45649017e-03,  5.37373349e-02,
                4.52498859e-03, -1.61806736e-02,  1.54691869e-02,
                4.36530188e-02,  8.42093974e-02,  2.12720297e-02,
                3.65959965e-02, -1.20446561e-02,  4.47615683e-02,
               -5.82285337e-02, -4.69586672e-03, -3.73253971e-02,
                4.55098860e-02, -1.75471175e-02, -1.39201889e-02,
               -1.25587964e-02,  1.23328473e-02,  2.16122083e-02,
                1.45873893e-02, -1.92219839e-02,  3.94796543e-02,
               -3.57443467e-03, -4.21496853e-02,  1.06691597e-02,
                2.64773741e-02, -1.34681650e-02, -1.02623485e-01,
               -4.36262973e-02,  3.62484530e-02, -4.82646525e-02,
                7.80529389e-03, -5.95573001e-02, -5.70402071e-02,
               -6.60517663e-02,  6.52047917e-02,  1.80767421e-02,
               -2.31865346e-02, -8.46131891e-02,  2.78796442e-02,
                1.37531450e-02,  4.75810096e-02, -2.76084226e-02,
                3.11912764e-02, -3.93198356e-02, -4.37822565e-02,
                5.91558591e-02, -6.12159669e-02,  4.90970686e-02,
               -4.56747087e-03,  3.18446197e-02,  1.98702179e-02,
                2.15840079e-02,  1.84831172e-02,  2.02920064e-02,
               -1.85871739e-02, -1.54211530e-02, -7.97762424e-02,
               -2.21173912e-02,  5.00475615e-03,  4.18125391e-02,
                3.54722887e-02,  2.66418494e-02, -3.64059844e-04,
                1.19073391e-02,  6.98634088e-02,  2.30949614e-02,
```

Figure 4 – Fragment of 128 numeric values of «face signature»

Features the first program that allows you to create a database from stock images of faces. This program is *1Signature.ipynb*. Initially, a directory with images of faces that are included in the database is needed. It called «*photosforfacenet*». It contains an image without a face to understand that if there is no face, the value «unknown» should be output. The number of face images depends on the computing resources of the workstation. The face image should be in the format *"person's name". png*, because during face recognition the name of the person is taken from the name of the image peculiar to this person. Running the program is accompanied by code to load libraries such as *os*, *PIL*, *numpy*, *matplotlib*, *pickle*, *cv2*, and *keras_models*. Libraries are necessary to support the codes in a program. The *os* library provides functionality in working with the operating system to retrieve information about files in a directory. *PIL* and *OpenCV* are Python image libraries used for image processing in Python. The *OpenCV* library is needed to retrieve the image from the camera. The next step is to load the Haar cascade classifier and FaceNET. Figure 5 shows the code responsible for loading the Haar cascade classifier and FaceNET [6].

```
HaarCascade = cv2.CascadeClassifier(cv2.samples.findFile(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml'))
MyFaceNet = FaceNet()
```

Figure 5 – Haar and FaceNET cascading classifier download code

The *haarcascade_frontalface_default.xml* file contains a pre-trained Haar classifier for detecting faces in the frontal position. This file is downloaded from *github.com* and should be saved in the directory with all files. All program files are located in C:\Users\Admin\Documents\machinelearn\Face-Recognition-with-FaceNET-main. In addition, you need to download the file *facenet_keras.h5* also via *github.com*. After all the necessary models have been loaded, it is necessary to proceed to the main code. The directory with photos is assigned a variable, which is specified as an empty database. Next, a for loop is used to repeat iterations equal to the number of files in the *photosforfacenet* directory. For each iteration a name is required, and it is taken from the name of the file located in the *photosforfacenet* directory. The above processes used the *OpenCV* library, but the image codes that are in the code apply the *PIL* format. The image layers in *OpenCV* are composed of *BGR* and *PIL* is composed of *RGB*. For this reason, the image is converted, turned into an array and a cropping process is performed based on x1, y1, x2 and y2. In the end, only the face image should remain. The name of the converted image is *gbr*. The face is then converted to an array, followed by resizing the face 160*160 and another conversion to an array. The image is then transmitted to FaceNET. As a result, a signature is created and stored in a «database» dictionary. Figures 6 and 7 show code fragments with detailed explanation of the code lines.

Next, you need to save this database with face signatures from this *photosforfacenet* directory to a file named *data.pkl*. This file is saved in the directory where the other files are located. For this purpose, the code shown in Figure 8 is applied.

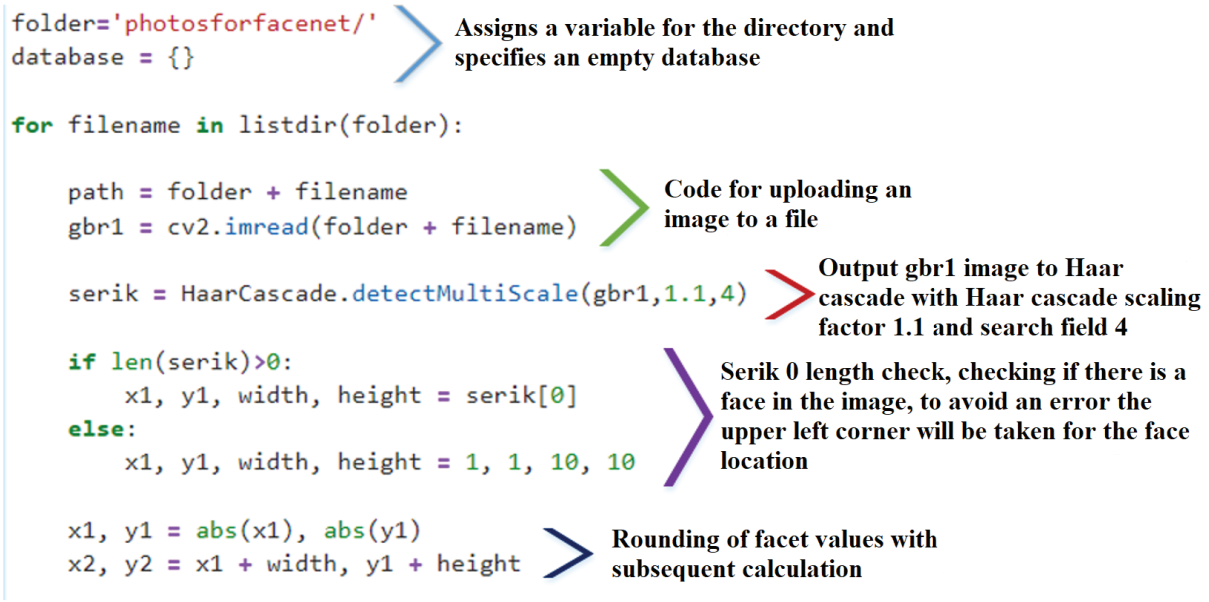

Figure 6 – The first part of the code fragment of the 1Signature.ipynb program with comments

```
gbr = cv2.cvtColor(gbr1, cv2.COLOR_BGR2RGB)
gbr = Image.fromarray(gbr)                    #Converting OpenCV to PIL
gbr_array = asarray(gbr)

face = gbr_array[y1:y2, x1:x2]

face = Image.fromarray(face)
face = face.resize((160,160))
face = asarray(face)

face = expand_dims(face, axis=0)
signature = MyFaceNet.embeddings(face)
database[os.path.splitext(filename)[0]]=signature
```

Convert the face to an array and then resize the face to 160*160 pixels and another conversion to an array

Transmit normalized image to FaceNET

Creating a signature

Defining the key of each signature with file name separation

Figure 7 – The second part of the code fragment of the 1Signature.ipynb program with comments

```
myfile = open("data.pkl", "wb")
pickle.dump(database, myfile)
myfile.close()
```

Figure 8 – Code for saving the data.pkl file

The next step is to create a second program identical to the first one, but with some changes. In the second program, an algorithm for recognizing a face taken from a camera image is developed. The number of libraries used is the same, code for loading the Haar cascade classifier and FaceNET is added. The next step is to add code to read the database from the *data.pkl* file. The standard *pickle* library has been applied for the file loading processes. Since the image is taken from the workstation camera, it is necessary to define the camera in the code. The OpenCV library and the variable cap = *cv2.VideoCapture(0)*, where 0 is the camera used. By default, the built-in camera is labeled with the number 0. If there is another camera, you can connect another camera by changing the value of 0 to another. Figure 9 shows the first part of the main code of the second program [7].

This code runs until the *Esc* key is pressed to terminate the program. The initial step is to capture frames from the camera, then the next line of code is similar to the line of code from the previous program, which is responsible for loading the Haar cascade image for face detection. The face is detected, face location is determined, conversion to PIL format is done, face-cropping operation is performed and conversion to 160*160 pixels size is done. Figure 10 shows the second part of the second program [8].

To determine if the received signature from the camera matches the signature from the database, it is represented that the norm value between «value» and «signature» will be 100, thus the loop will iteratively read the signature in the «values» database equal to the number of signatures in the «signatures» database. The face identifier «*identity*» will be left blank. This face identifier is necessary for displaying on the video in the window. The value «*key*» in the code corresponds to the person's name in the database dictionary, and in each iteration process, «key» and «value» will be read. The next line of code is responsible for calculating the norm, the difference between «value» and «signature» and its variable named as «dist». In case the value of dist is less than 100, it will be recorded as a possible match. «Key» will be written as «identity». The process will repeat until the second signature in the database and so on. This process will look through all the signatures in the database to find a better matching signature. When a matching signature is found, that is the shortest distance, «*identity*» will display the person's face. The next step is to display a rectangle around the person's face and their name [9].

```
cap = cv2.VideoCapture(0)

while(1):
    _, gbr1 = cap.read()

    serik = HaarCascade.detectMultiScale(gbr1,1.1,4)

    if len(serik)>0:
        x1, y1, width, height = serik[0]
    else:
        x1, y1, width, height = 1, 1, 10, 10

    x1, y1 = abs(x1), abs(y1)
    x2, y2 = x1 + width, y1 + height


    gbr = cv2.cvtColor(gbr1, cv2.COLOR_BGR2RGB)
    gbr = Image.fromarray(gbr)
    gbr_array = asarray(gbr)

    face = gbr_array[y1:y2, x1:x2]

    face = Image.fromarray(face)
    face = face.resize((160,160))
    face = asarray(face)
```

Defining a camera in the code

Capturing footage from the camera

Output gbr1 image to Haar cascade with Haar cascade scaling factor 1.1 and search field 4

Serik 0 length check, checking if there is a face in the image, to avoid an error the upper left corner will be taken for the face location

Rounding of facet values with subsequent calculation

 # Converting OpenCV image to PIL

Converting a face into an array and then resizing the face to 160*160 pixels and another conversion to an array

Figure 9 – The first part of the main code

```
face = expand_dims(face, axis=0)
signature = MyFaceNet.embeddings(face)

min_dist=100
identity=' '
for key, value in database.items() :
    dist = np.linalg.norm(value-signature)
    if dist < min_dist:
        min_dist = dist
        identity = key

cv2.putText(gbr1,identity, (100,100),cv2.FONT_HERSHEY_SIMPLEX, 1, (70, 255, 50), 2, cv2.LINE_AA)
cv2.rectangle(gbr1,(x1,y1),(x2,y2), (0,255,0), 2)

cv2.imshow('Raspoznavanie FaceNET',gbr1)

k = cv2.waitKey(5) & 0xFF
if k == 27:
    break

cv2.destroyAllWindows()
cap.release()
```

Transmitting a normalized image to FaceNET

Search for the closest face in the database for the detected face in the image

Output text and rectangle on video frame

The name of the window in which the frame will be displayed; gbrl: video frame.

Executing the program before pressing the Esc key

cv2.destroyAllWindows() closes all windows that were created during program execution using OpenCV;
cap. release() releases the resources occupied by the v2.Video Capture instance.

Figure 10 – The second part of the program AitzhanovSerik.ipynb

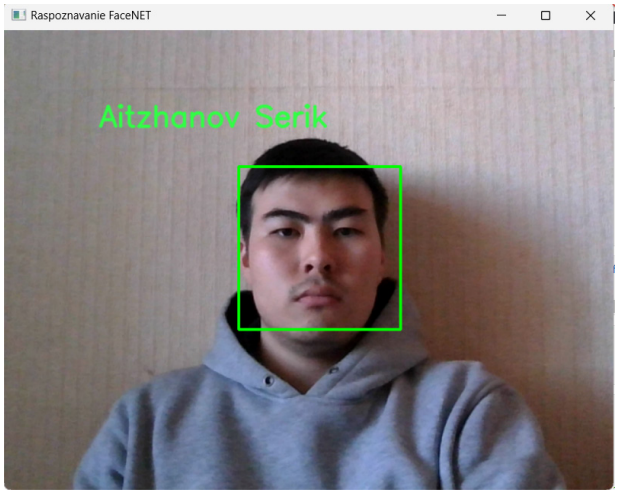

Figure 11 – Result of program execution

Final lines of code. The *cv2.putText()* function allows you to output text per video frame and takes a number of arguments. The frame of the video on which the text is output is labeled as *gbr1*, and the *identity* outputs the text in the frame. The *cv2.rectangle()* function allows you to output a rectangle per video frame. The *cv2.imshow()* function displays the current video frame on the screen. And the last function *cv2.waitKey()* waits for a keyboard key to be pressed and returns its code. If the *Esc* key is pressed, the loop is terminated. Figure 11 shows the result of executing this program [10].

3. Conclusion

The FaceNET model considered as an example is one of the most efficient and accurate models for face recognition based on deep learning. The capabilities of the model where a single photo taken by a camera is applied in the system are repeatedly considered. As the number of single face images increases, the accuracy rate increases. This study demonstrates the working principle of FaceNET model in practice using convolutional neural network technology to extract facial features and convert them into numerical vectors for comparison and identification of individuals by face.

Funding

This study was funded by the Ministry of Science and higher education of the Republic of Kazakhstan, Grant No. AP19678000.

References

1. Anderson R. Security engineering: a guide to building dependable distributed systems. – John Wiley & Sons, 2020.
2. Tsutsui S. (ed.). Intelligent biometric techniques in fingerprint and face recognition. – Routledge, 2022.
3. Aditya Mittal. «Haar Cascades, Explained» Medium, https://medium.com/analytics-vidhya/haar-cascades-explained-38210e57970d. Published December 21, 2020. Accessed April 11, 2023.
4. Schroff F., Kalenichenko D., Philbin J. Facenet: A unified embedding for face recognition and clustering //Proceedings of the IEEE conference on computer vision and pattern recognition. – 2015. – С. 815-823.
5. Shetty A. B. et al. Facial recognition using Haar cascade and LBP classifiers //Global Transitions Proceedings. – 2021. – Т. 2. – №. 2. – С. 330-335.
6. Viola P., Jones M. Rapid object detection using a boosted cascade of simple features //Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001. – Ieee, 2001. – Т. 1. – С. I-I.
7. E Woods R., C Gonzalez R. Digital image processing. – 2008.
8. Géron A. Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow. – " O'Reilly Media, Inc.", 2022.
9. McKinney W. Python for data analysis. – " O'Reilly Media, Inc.", 2022.
10. Heaton J. Ian Goodfellow, Yoshua Bengio, and Aaron Courville: Deep learning: The MIT Press, 2016, 800 pp, ISBN: 0262035618 //Genetic programming and evolvable machines. – 2018. – Т. 19. – №. 1-2. – С. 305-307.