Personnel identification for access to military facilities

Biometrics System Project

Vincenzo Francesco ZERBO – Matr. 1937077
Federico LAGRASTA – Matr. 1907104

Content table

Introduction

The purpose of this project is to develop a proof of concept of face-recognition software system that enables access control for personnel in the Italian Army barracks.

The system aims to enhance security measures by accurately identifying authorized individuals and preventing unauthorized access.

By leveraging computer vision techniques and machine learning algorithms, the software will detect and recognize faces in real-time, providing an efficient and secure access control mechanism.

Technologies and Libraries

The project utilizes the following technologies and libraries:

OpenCV: OpenCV (Open Source Computer Vision Library) is an open-source computer vision and machine learning software library. It provides a wide range of algorithms and functions to handle image and video analysis tasks, including face detection, recognition, and tracking.

Python: Python is a high-level programming language known for its simplicity and readability. It offers extensive libraries and frameworks for scientific computing, machine learning, and image processing.

Pillow: Pillow is a powerful image processing library for Python. It provides various functions for loading, saving, and manipulating images. In this project, Pillow is used for preprocessing and manipulating the face images.

CMAKE: CMake is a cross-platform build system used to manage the build process of the project. It simplifies the compilation and configuration of the software across different platforms and operating systems.

Haar Cascade classifier: The Haar Cascade classifier is a machine learning-based object detection algorithm. It is particularly effective for face detection, as it utilizes a set of pre-trained classifiers to identify facial features. In this project, the Haar Cascade classifier is used for initial face detection.

Numpy: Numpy is a fundamental package for scientific computing with Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions. Numpy is used for data manipulation and efficient storage of face images and features.

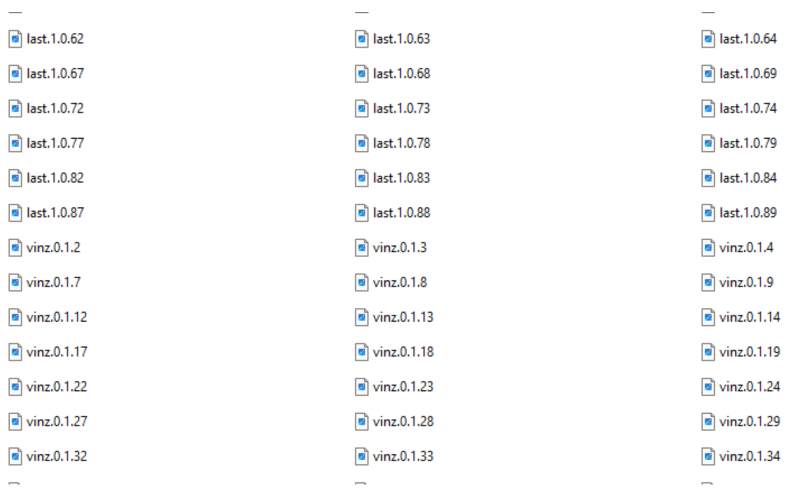
Modules

The project consists of several segmentation modules, each serving a specific purpose:

01_dataset

The script:

- captures images of faces using a webcam and saves them in the "dataset" folder
- initializes the webcam capture by creating a VideoCapture object. It sets the frame width and height to 640x480 pixels.
- loads the Haar Cascade classifier XML file for face detection. The XML file contains pre-trained classifiers to detect frontal faces.
- prompts the user to input the **last name of the user**, **authorization status** (1 for authorized, 0 for unauthorized), and an **incremental identifier**. **These inputs will be used to generate unique filenames for the captured images**.
- initializes a count variable to keep track of the number of captured images and displays a message to the user.
- enters a loop to continuously capture frames from the webcam until 90 images are captured. It reads a frame from the webcam, converts it to grayscale, and detects faces using the Haar Cascade classifier.
- Finally, the script prints a success message and releases the webcam capture. It also closes any open windows used for displaying the captured images.



```
last.1.0.62    last.1.0.63    last.1.0.64
last.1.0.67    last.1.0.68    last.1.0.69
last.1.0.72    last.1.0.73    last.1.0.74
last.1.0.77    last.1.0.78    last.1.0.79
last.1.0.82    last.1.0.83    last.1.0.84
last.1.0.87    last.1.0.88    last.1.0.89
vinz.0.1.2     vinz.0.1.3     vinz.0.1.4
vinz.0.1.7     vinz.0.1.8     vinz.0.1.9
vinz.0.1.12    vinz.0.1.13    vinz.0.1.14
vinz.0.1.17    vinz.0.1.18    vinz.0.1.19
vinz.0.1.22    vinz.0.1.23    vinz.0.1.24
vinz.0.1.27    vinz.0.1.28    vinz.0.1.29
vinz.0.1.32    vinz.0.1.33    vinz.0.1.34
```

02_training

The second script is responsible for training a face recognition model using the **LBPH (Local Binary Patterns Histograms)** algorithm. It creates an instance of the LBPH face recognizer and loads the **Haar Cascade classifier** for face detection.

The **getImagesAndLabels()** function is defined to retrieve the images from the dataset directory.

It iterates over the images, converts them to grayscale, and extracts the face regions using the Haar Cascade classifier; then, it collects the face samples and their corresponding subject IDs, obtained from the image filenames.

After collecting the face samples and subject IDs, the script proceeds to train the recognizer using the **train()** method. The face samples and subject IDs are passed as arguments to the training function.

The trained model is then saved to a **YAML** file using the **write()** method of the recognizer object. The file is stored in the "**trainer**" directory.

In summary, this script automates the process of training a face recognition model using the **LBPH** algorithm by collecting face samples from a dataset, associating them with subject IDs, and training the recognizer.

The trained model will be used for face recognition tasks, enabling the identification of individuals from new face images.

03_recognition

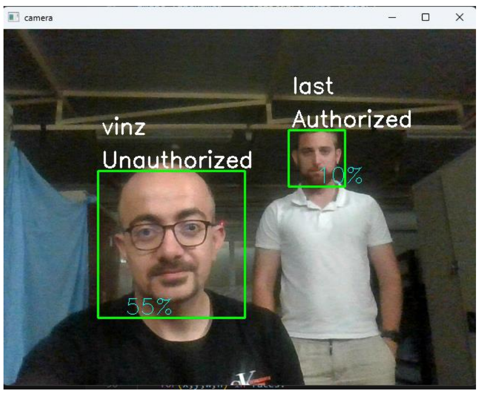
The third script implements the face recognition functionality using the **trained** LBPH (Local Binary Patterns Histograms) model. It begins by defining a function called **ResolveNameFromId()** which resolves the name of a person given their identifier.

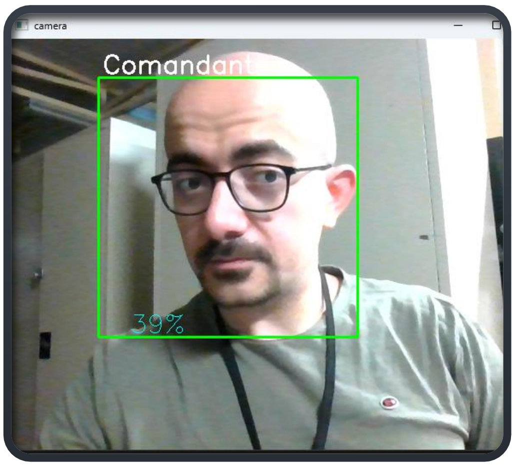
This function searches for the image file with the corresponding identifier and extracts the name from the filename.

Next, the script loads the trained **LBPH recognizer** from the **YAML** file using the **read()** method. It also initializes the **Haar Cascade classifier** for face detection. Then it defines variables and dictionaries to handle authorization levels and build a database of authorized personnel from the image dataset.

The script starts capturing video from the webcam using **OpenCV's VideoCapture** class. Inside the main loop, it detects faces in the captured frames using the **Haar Cascade classifier**. For each detected face, it predicts the **identity using the trained recognizer** and calculates the **confidence level**.

If the confidence level is below a certain threshold (**60** in this case), the script retrieves the authorization level, calculates the confidence percentage, and displays the **subject's name**, **authorization level**, and **confidence on the frame**. If the confidence level is higher, indicating an unknown face, it displays "**Unknown**" as the name.





04_test_dataset

This script is similar to the first but will have the purpose of collecting a set of faces that will be used to verify the accuracy of the recognizer during next phases of test, based on the training previously performed and the comparison with the provided dataset.

05_testing

The last script performs two tests to evaluate the performance of the trained AI model for face recognition. It calculates the **False Acceptance Rate (FAR)**, **False Rejection Rate (FRR)**, and **Equal Error Rate (ERR)** using **two** different test datasets, that include both the same people from the **training set** and **different other unknown faces** from an **open-source dataset** **(http://vis-www.cs.umass.edu/lfw/)**

It first loads the **trained face recognizer** using the LBPHFaceRecognizer from OpenCV and the **Haar Cascade classifier** for face detection.

Next, we have two functions defined: **CalculateFAR** and **CalculateFRR**.

These functions take an image folder as input (**test_dataset** and **test_dataset2**) and perform the evaluation for **FAR** and **FRR**.

For FAR calculation, it skips images with authorization status other than 0 (unauthorized), and for FRR calculation, it skips images with authorization status other than 1 (authorized).

The faces in the images are detected using the Haar Cascade classifier.

For each detected face, the face region is passed to the recognizer to predict the authorization. If the predicted authorization differs from the actual authorization, it increments the false positives (for FAR) or false negatives (for FRR) count.

In the main part of the script, it executes the first test by calling CalculateFAR and CalculateFRR functions with the 'test_dataset' folder. It prints the FAR, FRR, and ERR results. Then, it proceeds to the second test using the 'test_dataset2' folder and prints the corresponding rates.

Overall, this script provides a way to evaluate the performance of the face recognition system by measuring FAR, FRR, and ERR using test datasets. It helps assess the effectiveness of the trained AI model in distinguishing authorized and unauthorized individuals.

Evaluation

The results of the first test using the 'test_dataset' are as follows:

- **False Acceptance Rate (FAR)**: 0.4559 (45.59%)
- **False Rejection Rate (FRR)**: 0.9468 (94.68%)
- **Equal Error Rate (ERR):** 0.7013 (70.13%)

These results indicate that the face recognition system has a high false rejection rate, meaning it incorrectly rejects authorized individuals. The false acceptance rate is also relatively high, indicating that the system incorrectly accepts unauthorized individuals.

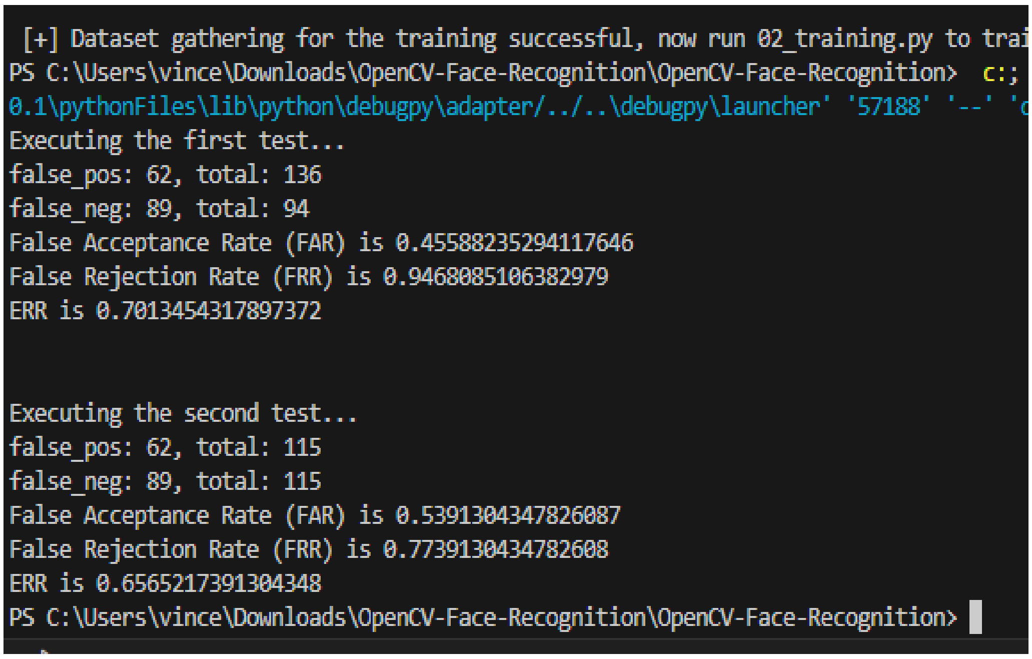
The results of the second test using the 'test_dataset2' are as follows:

- **False Acceptance Rate (FAR): 0.5391 (53.91%)**
- **False Rejection Rate (FRR): 0.7739 (77.39%)**
- **Equal Error Rate (ERR): 0.6565 (65.65%)**

$$FAR = \frac{False\ Acceptances}{Impostor\ Probes}$$

$$FRR = \frac{False\ Rejections}{Genuine\ Probes}$$

In the second test, the false acceptance rate and false rejection rate are still quite high, indicating a significant number of misclassifications. The ERR is **slightly lower** than in the first test, suggesting a slightly improved performance.



```
 [+] Dataset gathering for the training successful, now run 02_training.py to trai
PS C:\Users\vince\Downloads\OpenCV-Face-Recognition\OpenCV-Face-Recognition>  c:;
0.1\pythonFiles\lib\python\debugpy\adapter/../..\debugpy\launcher' '57188' '--' 'c
Executing the first test...
false_pos: 62, total: 136
false_neg: 89, total: 94
False Acceptance Rate (FAR) is 0.45588235294117646
False Rejection Rate (FRR) is 0.9468085106382979
ERR is 0.7013454317897372


Executing the second test...
false_pos: 62, total: 115
false_neg: 89, total: 115
False Acceptance Rate (FAR) is 0.5391304347826087
False Rejection Rate (FRR) is 0.7739130434782608
ERR is 0.6565217391304348
PS C:\Users\vince\Downloads\OpenCV-Face-Recognition\OpenCV-Face-Recognition>
```

These results highlight the limitations and challenges of the face recognition system implemented in the script.

The high false acceptance and rejection rates indicate that the system may struggle to accurately differentiate between authorized and unauthorized individuals.

Further improvements in the training data, model architecture, or the face recognition algorithm may be necessary to enhance the system's performance.

Future Use

The developed Face-Recognition software has potential applications beyond personnel access control in Italian army barracks.

It can be adapted for other security scenarios, such as airports, government facilities, and high-security areas.

Additionally, the system can be further enhanced with additional features, such as real-time monitoring, tracking, and integration with existing security systems. It can also benefit from ongoing research and advancements in machine learning and computer vision to improve accuracy and robustness.

Of course this is a proof of concept, that need, at least, to:

- Use a wider faces dataset with which to train the machine learning algorithm;
- Increase the accuracy of Gathering & Training mechanisms;
- Be Integrated with the Administration Software of the Italian Army;

Conclusion

The developed Face-Recognition software offers an effective solution for personnel access control in the Italian army barracks. By leveraging advanced computer vision techniques and machine learning algorithms, the system could detects and recognizes individuals, enhancing security measures.

With future improvements and evaluations, this technology can be applied to various security domains, contributing to safer environments.