SAPIENZA
UNIVERSITÀ DI ROMA

Biometric Systems a.y. 22-23

BiometricBites

Alessio Lucciola - lucciola.1823638@studenti.uniroma1.it
Danilo Corsi - corsi.1742375@studenti.uniroma1.it
Domiziano Scarcelli - scarcelli.1872664@studenti.uniroma1.it
Vasile Vladut Gorea - gorea.1695720@studenti.uniroma1.it

January 26, 2023

Contents

1 Introduction

For this final project, we decided to build a web application for access to the university cafeteria using facial recognition. There are two actors, students and administrators. Students can enter their reserved area with their credentials and enroll by entering their biometric data. When they show up at the cafeteria, the administrator (from their system) will identify the student, then obtain their identity (their data and the cost, which varies according to economic situation), and the student will access the cafeteria and pay what they are entitled to. For this reason, we are using an Identification Open-Set: open because even unregistered people can enter the cafeteria and, in the case of non-identification, they will pay the maximum price without any reduction. We designed the Frontend by using Typescript (with ReactJs) and the Backend by using Python (Django) and MySql for the DB.

2 Description of the application

We decided to build a web application to make the access to the university cafeteria easier and faster. At the moment, to access the cafeteria a QR code is needed. There exists an application that allows the user to generate a code that has to be shown before the payment. To log in, the user has to insert the SPID credentials so that the system can retrieve their data automatically (name, surname, economic condition, etc..). There are a few problems with the current approach: before going to the cafeteria the user has to be accepted by a human officer and the time required to receive approval is really long; the scanners used to check the QR code are slow and this creates long queues; the identity of the user is not verified when presenting the QR so the system can be easily spoofed.
For this reason, we decided to come up with a system that tries to solve the problems described so far. The idea was to create a system to manage the access to the cafeteria using face recognition. As we said in the introduction there are two actors:

- **Students**: They can access the application using their credentials (username and password). In their private area, they can make the enrollment by uploading new photos for face recognition, checking and deleting the uploaded photos, and checking their personal data (also how much they pay based on their economic condition)
- **Admin**: They can access the application using their credentials (username and password). In their private area, they can identify a user. Once a user is identified, their information is shown on the screen and the payment can be done according to how much they pay

A student can be:

- **Enrolled**: A student that uploaded some photos about their face and can access the cafeteria paying depending on the economic condition
- **Not enrolled**: A student that didn't upload the photos and didn't make the enrollment. They can still access the cafeteria but won't be identified and they are going to pay the maximum price without discounts

We assume that the credentials to access the system and provided by the Italian government using the SPID system. This way, we can immediately retrieve the information about a user such as name, surname, fiscal code, and the ISEE (the economic condition), and be sure that the data is legit. For simplicity, we didn't implement the data retrieval using the SPID system but we created a fake login system that works similarly to the SPID one, so once the user logs in, the system already has all the information about them.
When the student logs in, they are supposed to make the enrollment by uploading their photos. The idea is to take a few pictures with different poses in order to make the recognition more accurate. Once the enrollment is done, the user can go to the cafeteria and get identified. They will be asked to show their face and the admin will be shown their data and how much they pay. Notice that the student isn't required to prove their identity, so the system could generate some false matches. In general, this is an open set identification because the student doesn't claim an identity and they can't even be

enrolled. Of course, we don't have impostors and if the person isn't identified, they simply enter the cafeteria by paying the highest price.
In the next sections, we will explain more in detail what is the structure of the web application and show how the system actually works.

3 Structure

The application can be divided into **backend** and **frontend**. The frontend part is what the user sees and interacts with, while the backend part contains the logic of the application and it is composed by the web server and the database. It is a simple client-server application: the client can make a request to the server via a Rest API, the server receives the request and gives a response (possibly communicating with the database). Once the client receives a response, it is shown to the user.

3.1 Frontend

To implement the frontend, we decided to use Typescript in combination with ReactJS, a powerful open-source Javascript library for building user interfaces based on UI components. Each page of the web application is treated like a component, an entity that can be rendered to a particular element in the DOM using the React DOM library. The behavior of each component is managed by the states which are variables that can be dynamically changed and this will result in the rendering of the component. In order to create a multipage website, we used a routing library that allows the user to switch between pages. The style of the website was done using SCSS, a CSS extension that introduces variables, nesting, loops, and other features that allowed us to create the website in an easier way.
As mentioned before, the system has two actors: the student and the admin. Since their task is different, we divided the frontend structure into two parts. Once the user logs in, the system shows the right content based on the user's role. The student interface is composed of different pages that allow them to make the enrollment and check their personal information (see section 5). The admin interface is only composed of a single view in which there is a video stream that is connected to the recognition phase: when the user shows up, their information is shown on the screen and the admin can make the payment (see section 6).

3.2 Backend

To implement the backend we decided to use Django, a Python framework that allowed us to implement a fast and scalable web server. As already described, the communication between client and server is done using Rest APIs. When the server is running, it is always listening for a request from the client. Each request is done by specifying the URI of where the requested resource lies. The client sends an HTTP request to a certain URI, and the server will create a reply accordingly. For example "localhost:8000/api/get_user_info" is an API used to get the information of a certain user; the client sends an HTTP request to that URI specifying the id of a user as a parameter, and it gets an HTTP response with the requested data. Each API has a function that unpacks the request and creates a response. Django provides a view.py file that routes each URI to one of these functions. For example ("api/get_user_info" invokes a function "get_user_info" that contains the logic of the API that retrieves the information of a user).

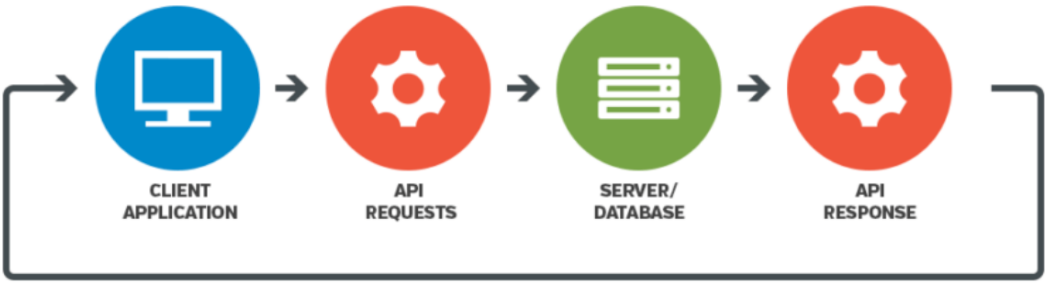

Figure 1: How an API works

We implemented a few APIs that are shortly described in this section:

- Login: It checks if the credentials provided by a user are valid and in this case, a new session is created. The information on the session is saved with cookies
- Get user info: It takes an ID in input and retrieves all the information about the user such as the name, surname, ISEE (the economic condition), and fiscal code
- Get attendance list: It retrieves the list of times in which the user accesses the cafeteria. Each entry contains the time and the amount paid
- Add attendance: It adds a new attendance to the cafeteria
- Get photo list: It retrieves all the photos of a user taken during the enrollment phase
- Delete photo: It deletes a photo of a user
- Upload photo: It adds a new set of photos of a user taken during the enrollment phase

The meaning of each API will be clearer in the next sections.
Each piece of information is saved to a database and retrieved when needed. We decided to use MySQL, an open-source relational database management system (RDBMS). The structure of the database is quite simple. It contains 3 tables:

- Users: It contains the login information such as the id of the user, the email address, the password and the role of the user
- User info: It contains the user's personal information such as the name, the surname, the fiscal code and the ISEE (the economic condition)
- User attendances: It contains the information on the users' attendances. Each attendance has an id, the id of the user, the date and the paid amount

The id of the user is the primary key of each table and it is used to link the tables.

3.2.1 Recognition on the backend with video streaming

Since the recognition module was implemented in Python in the system backend, we had to find a way to transfer the information about the person that has to be identified to the backend, in order to perform the recognition and return the useful information back to the front-end.

We decided to use WebSockets because they allow a persistent connection between the client and the server. The flow of the application is the following (drawing in Figure 14):

1. When the Admin connects to the client, it establishes a WebSocket connection to the server
2. The Admin webcam starts to capture the video stream. For each frame captured:
   (a) The single frame is captured by the administrator webcam on the administrator client;
   (b) The frame is converted in a `base64` string and sent to the server through the WebSocket;
   (c) The backend receives the encoded frame, and it decodes it from `base64` string to a usable format
   (d) The face is localized, the ROI is extracted and a box is drawn on the frame around the face;
   (e) If the face is present, then it's recognized with the biometric module, and an identity is returned, as well as the confidence;
   (f) The information (Image with a box around the face, recognized identity, confidence) is sent back to the front end through the WebSocket connection
   (g) On the front end, the received image is shown, as well as the cumulative confidence (because of the multi-biometric approach described in subsection 6.1) and the cumulative identity.
3. When the admin logs out, the WebSocket connection is closed.

For performance reasons, the recognition process is not done on every frame that the server receives, otherwise, the video stream would have a major delay due to the time needed for the recognition process for each frame. This would not allow the normal execution of the procedure. What we've done is perform the evaluation only for every `DELTA_RECOGNITION` frames.

This value was set to 5, meaning if the user's face is in the frame for 2 seconds, at 15 frames per second, 30 frames get sent to the server, but the recognition is performed only on $\frac{30}{\texttt{DELTA\_RECOGNITION}} = \frac{30}{5} = 6$ frames. The localization of the face and the drawing of the rectangle around it is performed for all the frames.

This allows the video stream to be fluid, and the recognition to still be reliable.

4 Models

We decided to test various approaches regarding face recognition, in order to see the variation of performances among different models and techniques. The approaches we decided to use are:

- Local Binary Patterns Histograms - LBPH
- Support Vector Machine Classifier - SVC
- VGG Face deep network

4.1 LBPH

This is a method for face recognition that is based on the analysis of the local binary patterns (LBP) in a certain image. This method works on grayscale binary images, so each image has to be converted into this format before the feature extraction. It is a simple yet very efficient texture operator which labels the pixels of an image by thresholding the neighborhood of each pixel and considers the result as a binary number. Using the LBP combined with histograms we can represent the face images with a simple data vector. The histogram is the feature vector of the image, and it's possible to have a measure of similarity (or distance) between two faces by just measuring the similarity (or distance) between two histograms. [1]. There are many similarity measures to use to achieve this task, after some experiments we decided to use Pearson's correlation coefficient. Usually, it's obtained via a Least-Squares fit and a value of 1 represents a perfect positive relationship, -1 a perfect negative relationship, and 0 indicates the absence of a relationship between variables. [2].

$$r = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2(y_i - \overline{y})^2}} \tag{1}$$

Where r is the correlation coefficient, x_i and y_i are the values of the x-variable and y-variable in a sample, $\bar{x}$ and $\bar{y}$ are the mean of the values of the x-variable and y-variable.

We used this approach in our system by taking advantage of OpenCV's LBPH classifier.

4.2 SVC

A support vector machine (SVM) is a supervised learning model that allows to perform binary classification. It works by fitting a hyperplane that best separates the data points between two classes. One reasonable choice as the best hyperplane is the one that represents the largest margin between two classes.

In order to perform multiclass classification with a support vector machine, it's possible to use the one-versus-one method, in which we build as many simple SVM classifiers as all the possible couple of classes, and then take the vote by using the max-wins-voting strategy. According to this strategy, every classifier assigns the instance to one of the two classes, and the class with the most votes determines the

instance classification [3]. In this case, the model outputs the probability for a template to belong to a certain class. It's not possible to use the model to extract the feature vector from the template.

We used this classifier by taking advantage of Python's Scikit Learn library.

4.3 VGG Face

The VGG Face is a deep convolutional neural network capable of recognizing faces. The model was developed by the Visual Geometry Group at the University of Oxford and was trained on a dataset of 2.6 million images with 2,622 different individuals [4]. We don't use the VGG Face model to have a direct prediction, but we use it as a feature extractor.

To compare two feature vectors, it's possible to use one of many distance measures. In this case, we decided to use cosine similarity, that's defined as follows:

$$cosine(A, B) = \frac{A \cdot B}{||A|| \cdot ||B||} \in [-1, 1] \tag{2}$$

Where A and B are the two feature vectors of the same length.

In order to use this classifier in our system, we took advantage of Python's "deepface" library.

5 Enrollment

5.1 Sign up and Log in

We assumed that users' data comes from the SPID system: We built the system assuming that DiSCo Lazio (the entity that takes care of the canteen service) would create an API in order to retrieve the data on a user, so Sign Up is necessary for access to the system.

When a user accesses the web application, if they're not logged in, a login page will show, requiring the email and password.

5.2 Upload photos

The enrollment process requires the user to upload a set of four photos, in different poses and expressions, in order for the face recognition system to be able to recognize the subject in different variations.

When the user accesses the web app, if there isn't a photo of them, they will be displayed a page where they are invited to upload their first photos.

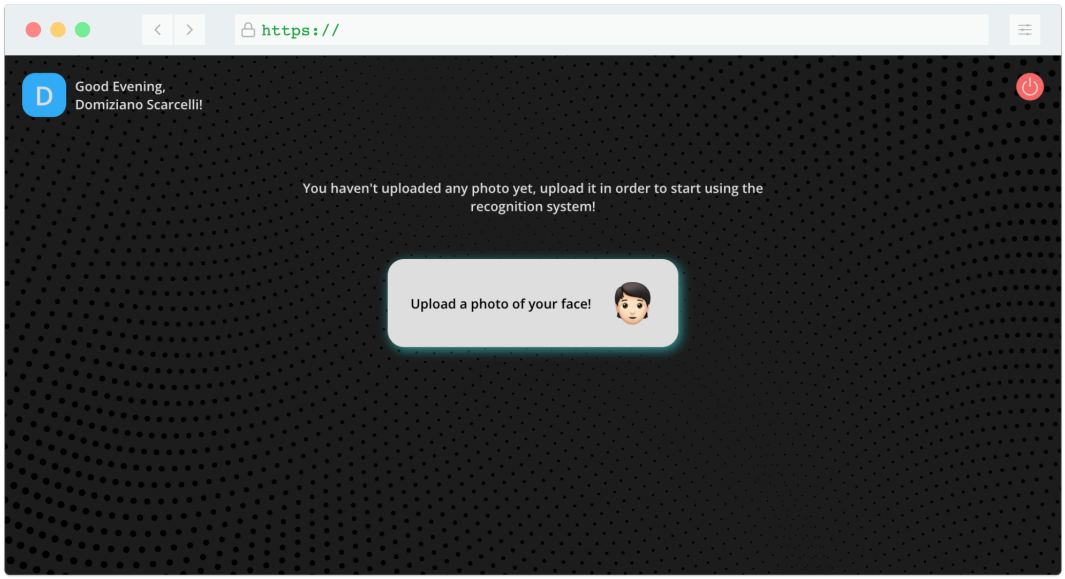

Figure 2: Not enrolled user homepage

The procedure is then guided: the user sees their webcam on the right, and on the left there are instructions on how to take the photo. The following photo variations will be asked in order:

1. Take a photo of your face in your neutral expression

2. Take a photo of your face while smiling
3. Take a photo on one of your sides
4. Take a photo on the opposite side

The user has to take a photo for each instruction, and each time they do that, the following instruction will appear. Once all photos are taken, the user can choose to upload them to the server or to retake them.

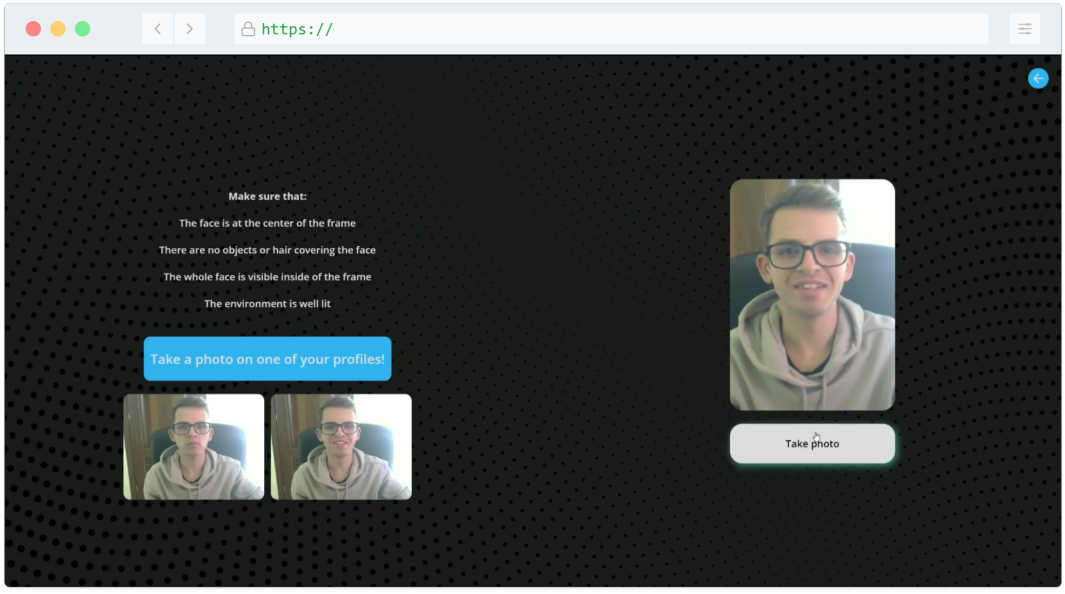

Figure 3: Interface during the enrollment: the user is asked to take few pictures with some variations

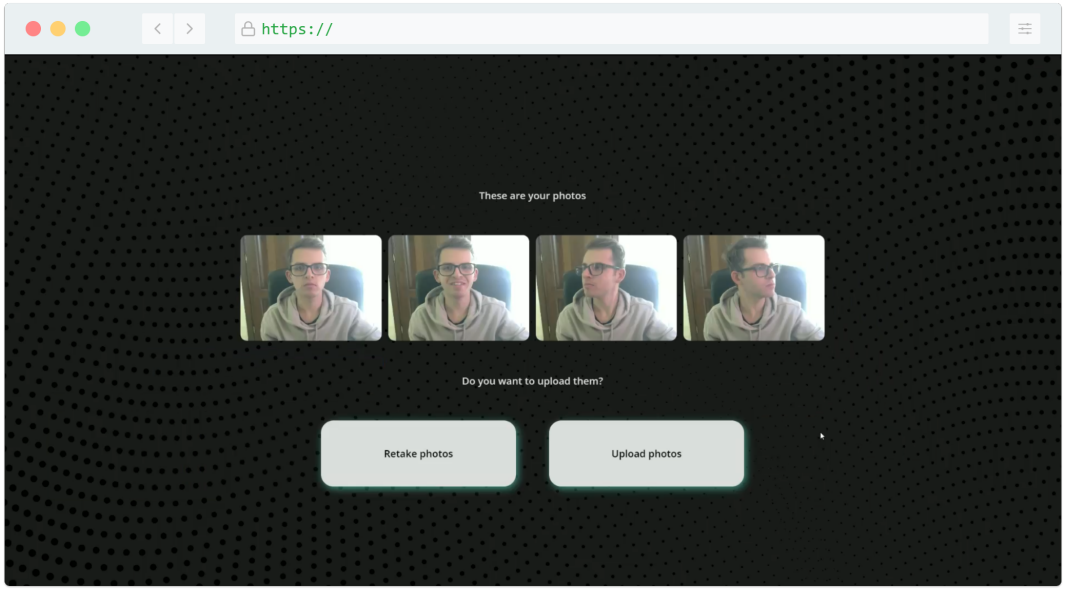

Figure 4: Interface when all photos have been taken

Note that the page uses a client-side face detector under the hood, that will prevent taking the photo if the face is not present, in order to avoid blank photos that are going to damage the gallery set.

5.3 Enrollment pipeline in the backend

The photos are uploaded to the server via an API call, which body contains the photo encoded in `base64` strings. They're then saved into a folder on the server, that models the gallery set and has the following structure:

```
\samples
    \1
        \img-0.jpg
        \img-1.jpg
    \2
        \img-1.jpg
        \img-2.jpg
        ...
    ...
```

Each folder inside "samples" is labeled after the user's unique ID, and inside each of those folders, there are the images that are uploaded by that specific user.

5.3.1 Image pre-processing

Once the photos are uploaded, the image pre-processing procedure is automatically triggered. First of all, there is the phase of face detection and localization inside the photo. This is done using Haar Cascade classifiers, both for frontal and side face detection. We assume that each photo contains one and only one face, so we first use the frontal face Haar cascade classifier to obtain the location of the face, if that fails to detect the face, then we use the side face classifier. If it also fails, the image is skipped. This occurs almost never since every uploaded image surely contains a face because of the frontend face detector in the enroll phase.

The localization process will output a series of numbers `x, y, w, h` that describe the initial coordinates of the point `(x,y)` where the face regions start, and the length of the horizontal `w` (width) and vertical side `h` (height).

Once we obtain these coordinates, we can crop the region of interest (ROI), in order to have a smaller image that only contains the subject's face region.

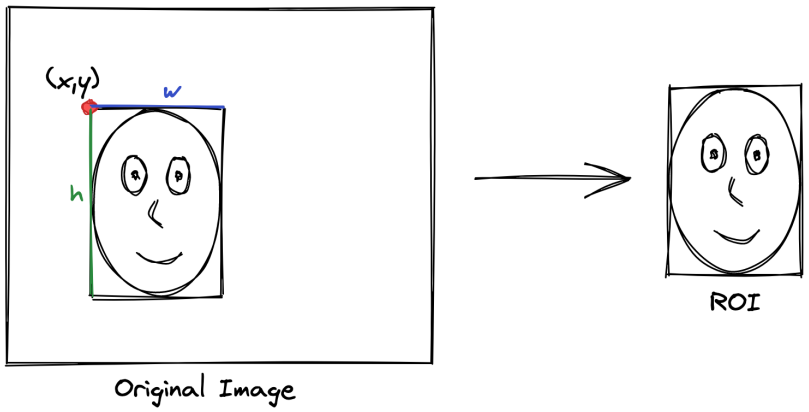

Figure 5: From original image to ROI

The original image is then replaced with the new processed image, and the `_processed` text is appended after the image file name in order to not process that image again whenever a new user will enroll and trigger the pre-processing pipeline in the future.

Each time the API call is made, and so the new enrollment photos are uploaded, the feature extraction and model training phase described in the next section is automatically triggered on the backend, in order to update the gallery set with the new photos and eventually train the model if it's needed.

5.3.2 Feature extraction and model training

Once the images have been processed we move on to the feature extraction phase. As we described earlier (section 4), in the entire project, we will use one of the three different types of classifiers, in

order to compare their recognition capabilities.

VGG Face and LBPH don't need training, because they're used to obtain the feature vector of each image, which will be compared to the future probes. SVC, on the other hand, needs training.

- **VGGFace**: Each image in the *samples* folder is scanned, and for each of them, we extract the feature vector using the VGG Face model. The feature vector is saved into an array of tuples that has the following structure:

  ```
  gallery = [(identity, feature_vector)]
  ```

  where each tuple has two elements: the identity of the template, and the feature vector extracted by the template. Once done that, we can save the file on the file system, and this is the system's gallery, which will be updated every time a new gallery template comes in.

- **LBPH**: As we saw earlier LBPH uses the feature vector of each image to perform recognition and has pretty much the same procedure as VGG Face model. To extract the features we scan each image into the **samples** folder, and for each of them:

  1. We extract the feature vector from the image, which in this case will be the image's histograms, and put it into a list (features).
  2. We extract the label associated with the image, and put it into another list (labels), at the same index of the feature vector.

  Once done, we can save the extracted feature vectors and the corresponding labels on the file system, and these will be the system's gallery, which will be updated every time a new gallery template comes in.

- **SVC**: Since SVC cannot be used to extract the feature vector of a template, we cannot build a data structure, like an array before, that can be our gallery. We need to train the model from the data we have, in order to let it be able to predict future probes. In order to do that, we scan each image in the **samples** folder, and for each of them:

  1. We extract the feature vector from the image, which in this case will be the image's histograms of gradients, and put it into the `X` list.
  2. We extract the label associated with the image, an put it into the `y` list, at the same index of the feature vector.

  We can then fit the `X`(feature vectors) and `y` (identity/ground truth) to the model, in order to obtain a trained model. The trained model is then saved, and we can consider the model as our gallery since it embeds information about all the enrollment images on which it has been trained, and will be able to predict future probes.

At this point, the enrollment process is terminated, from this moment the user can go to the cafeteria to be submitted for recognition by the cafeteria operator (admin).

5.4 User data management

Once the user ends the first enrollment, they will be shown another homepage as soon as they log in. On the homepage, they are given the possibility to add new photos, see their uploaded photos (the ones used in the previous enrollments), and check their information. There is also a box that shows the attendance records to the cafeteria with the amount paid and the date of entry. Every time a user makes the payment, a new record is added to the table.
The first button ("add other photos") takes the user to the enrollment phase as described in the previous section.

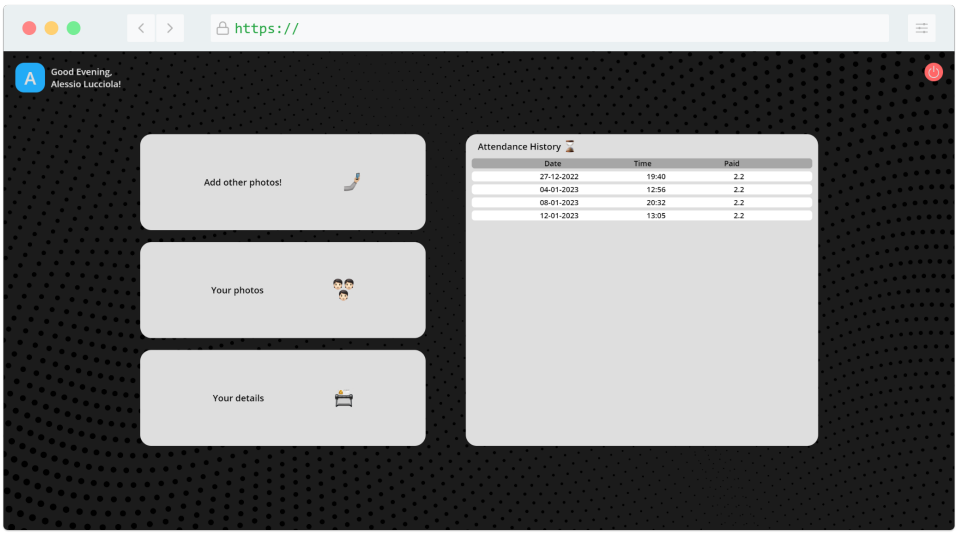

Figure 6: Enrolled user homepage

The second button ("your photos") opens a new page in which the user can see the photos that they uploaded during the enrollment phase. They also have the possibility to delete some of them by clicking the button under a photo.

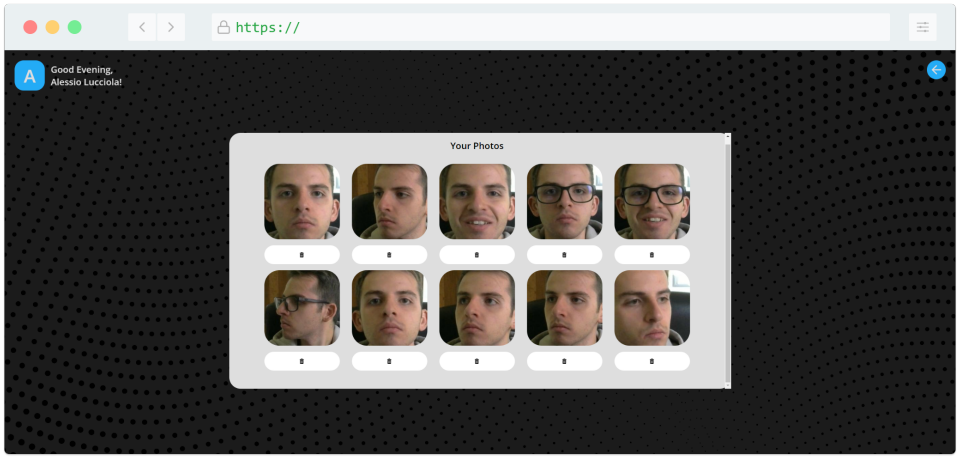

Figure 7: User photos interface

The third button ("your details") shows the user's information such as the name, surname, fiscal code, and how much the user pays to access the cafeteria. This information is supposed to be retrieved using some APIs provided by the Italian government. For our purpose, it is taken from the database we have created for this project. The cost is calculated dynamically depending on the economic condition of the user: the lower the ISEE, the lower the cost.

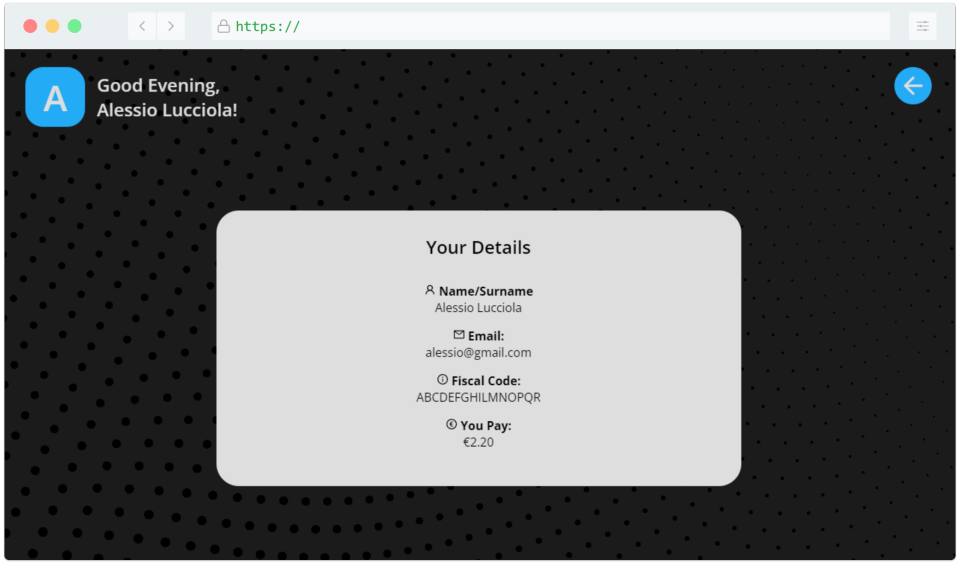

Figure 8: User information interface

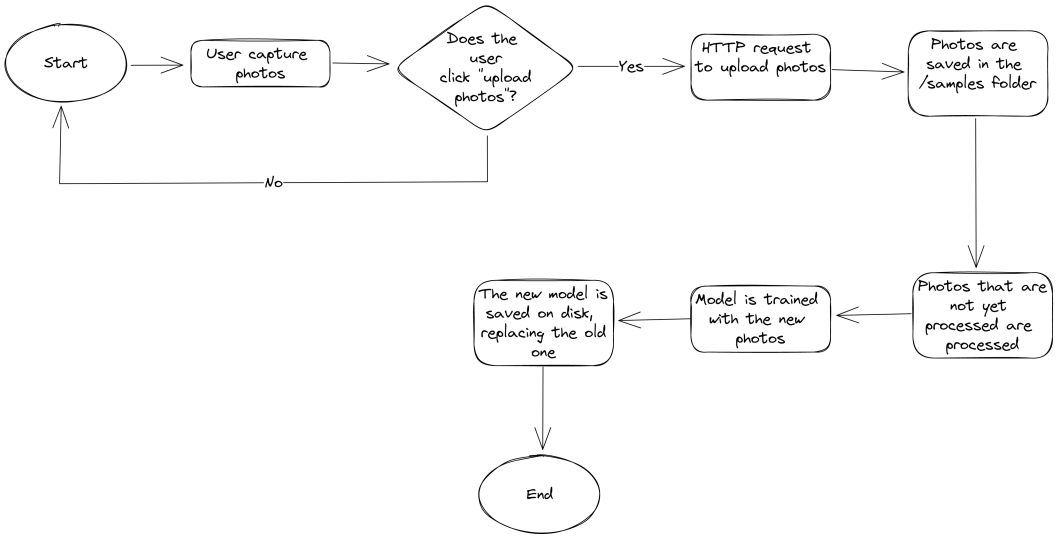

Figure 9: User enrollment flowchart

6 Recognition

In the recognition phase, we assume that the administrator has the computer connected to a camera that will be used to perform the recognition. They will be responsible for having the students correctly positioned in front of the camera to capture their faces and for confirming the recognition. It is also assumed that the administrator is already registered within the system always taking advantage of the data provided by DiSCo Lazio. After logging in, in the background, the trained model containing the student templates will be preloaded, and the connection to the camera will be initialized. At this point, the administrator will encourage the first student to be presented for recognition.

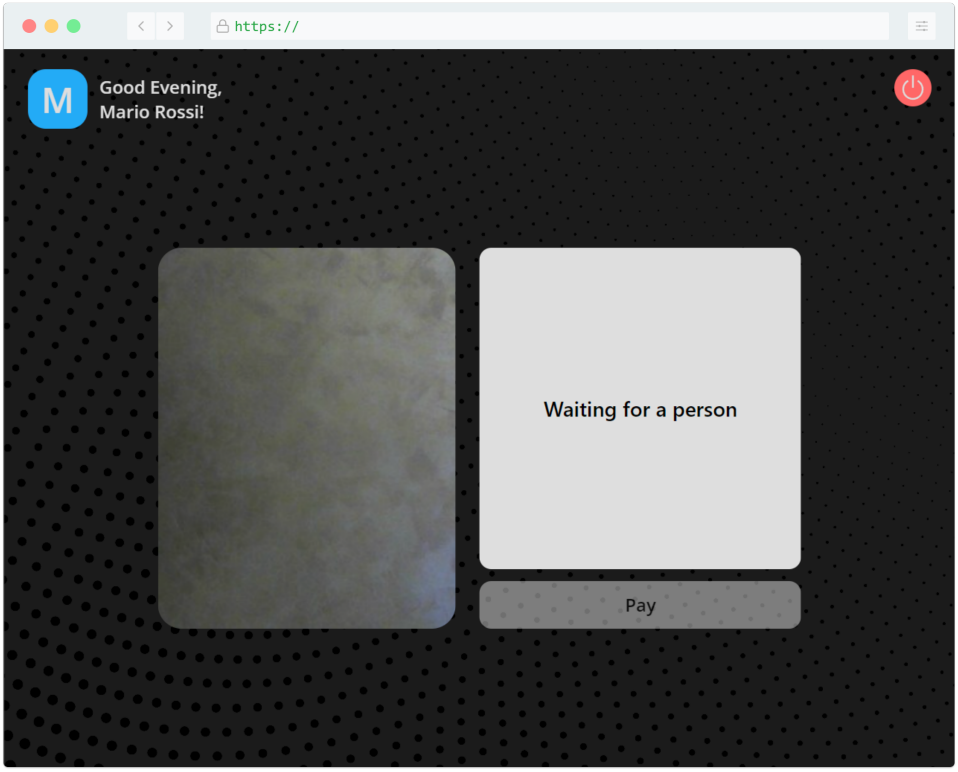

Figure 10: Admin homepage

As soon as the user enters the camera's field of view each frame will be sent to the model recognizer, which will extract the **Region of Interest (ROI)** containing the captured face, create an encoding of it, and compare it with those contained in the trained model. As soon as there is a match that meets a certain threshold then the actual identity of that student will be returned, including first name, last name, and price to pay to access the cafeteria.

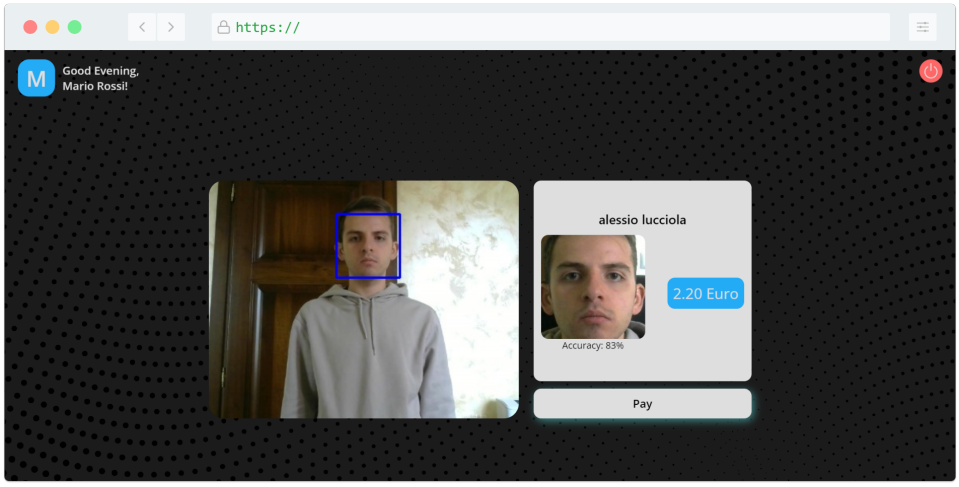

Figure 11: Admin interface when a student has been recognized

If there is a student or even a person who wants to access the cafeteria but has not enrolled, the system, after detecting that the encoding of the input probe does not match any of the templates in the gallery (by returning a confidence value below the threshold) will return "Unknown" as identity and the maximum price to pay to access the cafeteria.

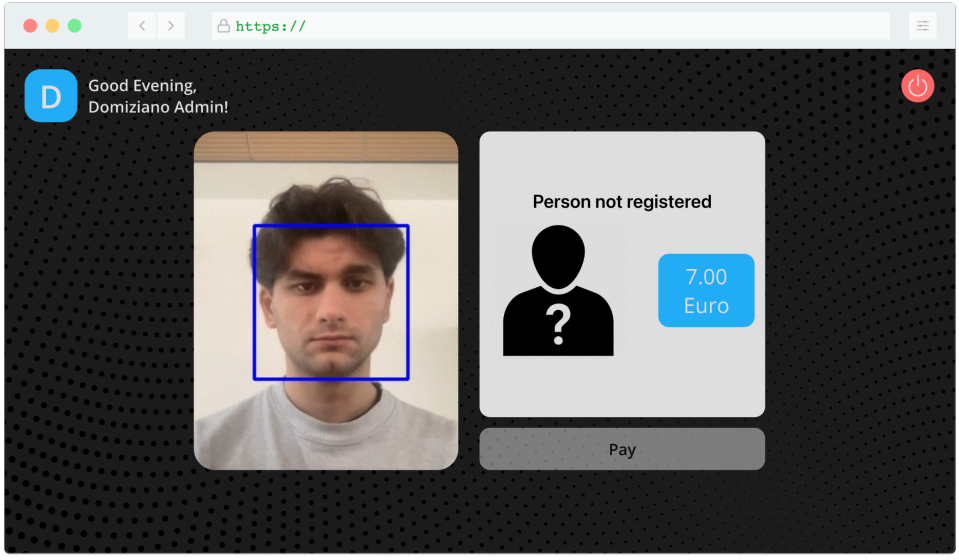

Figure 12: Admin interface when a student hasn't been recognized

After the system has returned the identity of the person in front of the camera, if the administrator has any doubts about it will make sure to confirm it and after that will accept the payment and proceed to identify the next student in the same manner as described above. If the admin clicks on the "pay" button, a new attendance record is added in the user's private area.

6.1 Multi-biometric approach

Since the system captures a video for each person that has to be identified, we have a set of frames in order to perform the recognition. Returning the result after evaluating just the first frame could be not so reliable, since the model may be wrong on a single instance. Our idea was to implement a multi-biometric approach that would consider the majority of results over all the frames acquired since the person enters the webcam frame. In a more detailed way, let $f_{id}^{(t)}$ be the frame of the person with id id at time t, let t_0 be the time when the person enters the webcam frame, and t_n the current time. Let also $predict(f)$ be the function that takes in input the frame f and returns an id as the predicted identity in the frame, then in the recognition phase, we build this array:

$$\text{predictions} = [predict(f_{id}^{(t)}) \quad \forall t \in [t_0, t_n]] \tag{3}$$

And select as the predicted identity the identity that appears most of the time in the "predictions" array.

When the person exits the webcam frame, then the array is emptied.

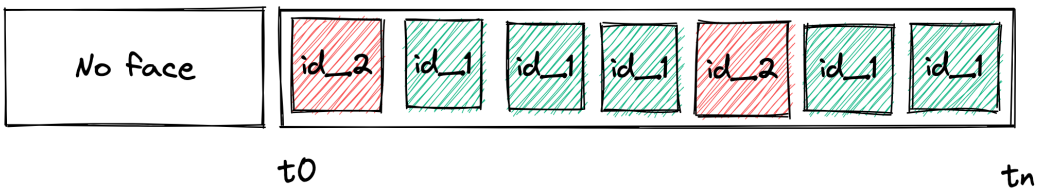

Figure 13: An example of the multi-biometric approach

We can see that in this example the model predicted id_2 two times and id_1 five times, so id_1 is the one returned, as it's more probable to be the correct prediction.

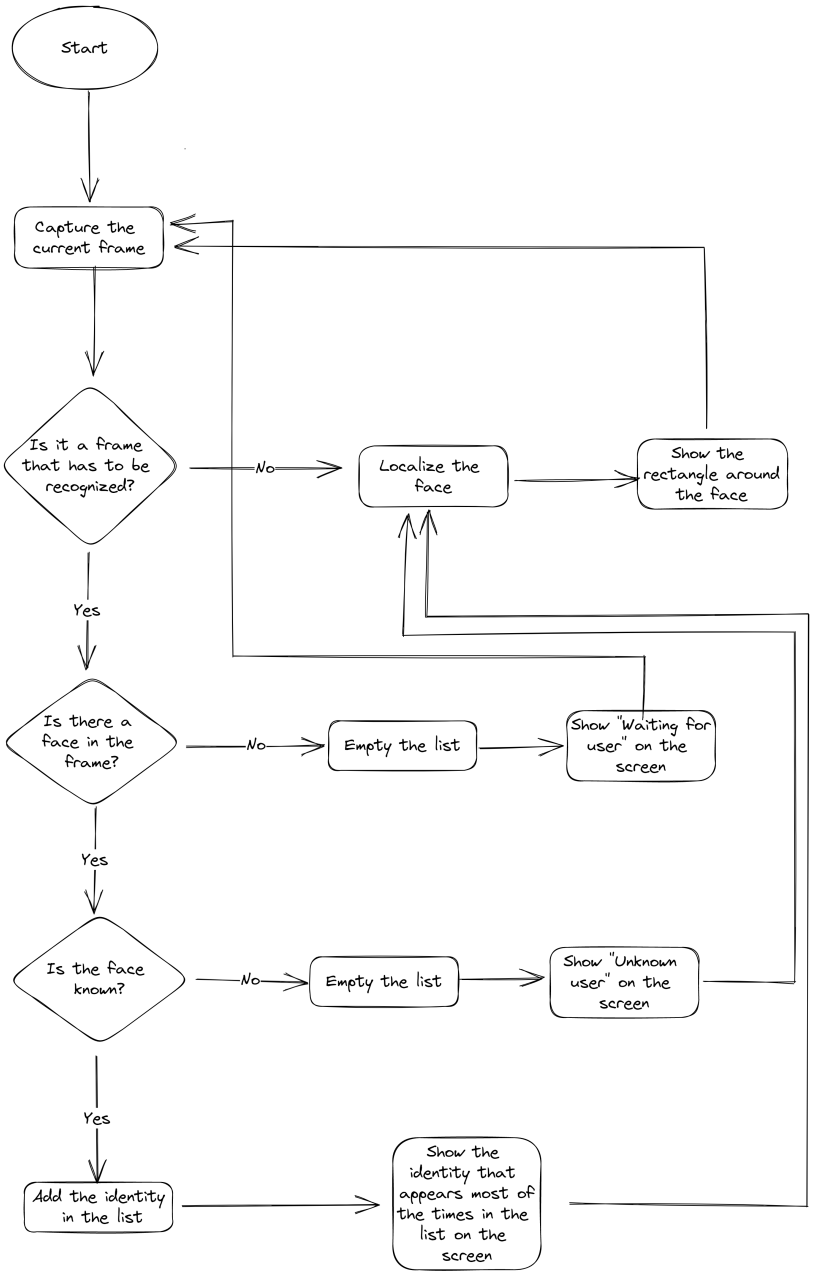

Figure 14: Admin recognition flowchart

7 Evaluation

Regarding the performance evaluation, we decided to use the all-probe-against-all-gallery approach. This decision was due mainly to use the same approach for all three of the methods. This because VGGFace and LBPH are used to extract feature vectors from the images, so it's possible for us to create a single array of feature vectors in order to perform all-against-all. On the other hand, this isn't possible for SVC, since the model cannot be used to extract features, and it needs training. Because of that, it's not possible to have a single array of feature vectors to use for the all-against-all, and so the division in gallery set and probe set is necessary.

Even if our system uses an open-identification approach, we decided to test our models even in a verification scenario meaning that a user can claim an identity. This was done only to make further tests and see how strong the models are.

The evaluation allowed us to retrieve useful metrics like the following where p_j in the j-th probe in the probe set, s_{xj} is the similarity between probe p_j and a template g_x in the gallery (where g_x is the best match meaning the template associated with a subject that is more similar to the probe p_j) and $id(template)$ is a function that, given a template, it outputs its associated identity. The metrics used in the verification operation are:

- **Genuine Acceptance Rate (GAR)**: Defines the number of correctly accepted probes, over the number of all the genuine probes

$$GAR(t) = \frac{|\{p_j : s_{xj} \geq t \wedge id(g_x) = id(p_j)\}|}{|\{p_j : id(g_x) = id(p_j)\}|} \; \forall p_j \in P_G \tag{4}$$

- **Genuine Rejection Rate (GRR)**: Defines the number of all correctly rejected probes, over the number of all the impostor probes

$$GRR(t) = \frac{|\{p_j : s_{xj} \leq t \wedge id(g_x) \neq id(p_j)\}|}{|\{p_j : id(g_x) \neq id(p_j)\}|} \; \forall p_j \in P_N \tag{5}$$

- **False Acceptance Rate (FAR)**: Defines the number of wrongly accepted probes, over the number of impostor probes

$$FAR(t) = \frac{|\{p_j : s_{xj} \geq t \wedge id(g_x) \neq id(p_j)\}|}{|\{p_j : id(g_x) \neq id(p_j)\}|} \; \forall p_j \in P_G \cup P_N \tag{6}$$

  Notice that we consider the case in which the probe p_j can belong to an enrolled user or not. This is done in a verification scenario in which even an enrolled user can act as an impostor claiming a wrong identity.

- **False Rejection Rate (FRR)**: Defines the number of wrongly rejected probes, over the number of all genuine probes

$$FRR(t) = \frac{|\{p_j : s_{xj} \leq t \wedge id(g_x) = id(p_j)\}|}{|\{p_j : id(g_x) = id(p_j)\}|} \; \forall p_j \in P_G \tag{7}$$

The metrics used in the (open-set) identification are:

- **Detection and Identification at rank k (DIR(k))**: Defines the probability with which individuals, found in a database, are correctly identified in an open-set identification, within rank k (the correct subject is returned to position k), given by the ratio between the number of individuals correctly recognized within the rank k over the number of all genuine probes

$$DIR(t,k) = \frac{|\{p_j : rank(p_j) \leq k \wedge s_{xj} \geq t \wedge id(g_i) = id(p_j)\}|}{|P_G|} \; \forall p_j \in P_G \tag{8}$$

- **False Rejection Rate (FRR)**: Defines the number of wrongly rejected probes, over the number of all genuine probes. In the open-set identification it matches the number of times the user is wrongly identified over the number of all genuine probes

$$FRR(t) = 1 - DIR(t,1) \tag{9}$$

- **False Acceptance Rate (FAR)**: Defines the number of wrongly accepted probes, over the number of impostor probes. In the open-set identification we only consider that we have a probe not in the gallery for which the highest similarity achieved is above the similarity threshold

$$FAR(t) = \frac{|\{p_j : max_i \; s_{ij} \geq t\}|}{|P_N|} \; \forall p_j \in P_N \; \forall g_i \in G \tag{10}$$

- **Genuine Rejection Rate (GRR)**: Defines the number of all correctly rejected probes, over the number of all the impostor probes. In the open-set identification we only consider that we have a probe not in the gallery for which the highest similarity achieved is below the similarity threshold

$$GRR(t) = \frac{|\{p_j : max_i \; s_{ij} \leq t\}|}{|P_N|} \; \forall p_j \in P_N \; \forall g_i \in G \tag{11}$$

All those metrics depend on the threshold, so we ran the algorithm for every possible threshold between 0 and 1 with an increment of 0.01, in order to have a high details view of the performances. We set the threshold between 0 and 1 because we worked with the normalized similarities, meaning similarities between templates that are defined in the $[0, 1]$ interval.

7.1 Datasets

We used two public datasets in order to carry on a performance evaluation that was more reliable as possible. The datasets are:

- Labeled Faces in the Wild (LFW)
- Olivetti Faces

7.1.1 Labeled Faces in the Wild

The dataset contains more than 13,000 RGB images of faces collected from the web. Each face has been labeled with the name of the person pictured. 1680 of the people pictured have two or more distinct photos in the data set [5]. All the images are already cropped on the face region, which was detected by the creators of the dataset using the Viola-Jones face detector.

The images are also "funneled", meaning that the images in the dataset are aligned so that the faces are centered and oriented in the same direction.

Since our system requires at least 4 photos in different poses for each user to be enrolled, we decided to filter the dataset, using only the photos of the individuals that have at least 4 photos of them in the dataset, with a total number of photos of 6733 of 610 different subjects.

The dataset contains very exaggerated poses, expressions, and other variations that could make the model fail the recognition phase. Because of that, we also decided to use a simpler dataset, that is Olivetti Faces.

7.1.2 Olivetti Faces

The Olivetti faces dataset is a dataset of photographs of the faces of 40 individuals, taken at different times, with different expressions and lighting conditions. Each individual is represented by 10 images in the dataset, for a total of 400 images. The images are grayscale, with a resolution of 64x64 pixels. The dataset was created by AT&T Laboratories Cambridge in the late 1990s and it is considered one of the oldest and most used datasets for face recognition and facial expression detection. It has been used extensively in the past to evaluate and compare different algorithms and techniques for facial recognition [6].

This dataset is more suitable for our use case since the system will be deployed inside of a cafeteria, where the administrator will make sure that the user that has to be recognized is well visible in front of the camera.

Also in this case, the images in the dataset are already cropped with the face region, so no other face localization and ROI cropping were needed.

7.2 All-probes-against-all-gallery

7.2.1 Algorithm for the open set identification

```
genuine_attempts = 0
impostor_attempts = 0
GR = 0 #Genuine Rejections
FA = 0 #False Acceptances
for each row i: #for each probe
    genuine_attempts++
```

```
    impostor_attempts++

    label_i <- Identity of probe template i
    ordered_similarities <- Descending ordered list of tuples (label_j, similarity)
    best_match <- ordered_similarities[0][0] #Id of best match
    best_similarity <- ordered_similarities[0][1] #Highest similarity
    if (best_similarity >= threshold):
        if (label_i == best_match):
            DI[0]++
            for every column j: #for each template
                label_j <- Identity of gallery template j
                similarity <- Similarity between probe i and gallery template j
                if exists j such that label_i != label_j AND similarity >= threshold:
                        FA++
                    else:
                        GR++
        else:
            for every column j: #for each template
                label_j <- Identity of gallery template j
                similarity <- Similarity between probe i and gallery template j
                if exists j such that label_i == label_j AND similarity >= threshold:
                    DI[j]++
            FA++
    else:
        GR++

    DIR[0] = DI[0]/genuine_attempts
    FRR = 1 - DIR[0]
    FAR = FA/impostor_attempts
    GRR = GR/impostor_attempts
    for each k in [1, gallery_cardinality]:
        DIR[k] = (DI[k] / genuine_attempts) + DIR[k-1]
```

Listing 7.1: Open-Set Identification multiple template

7.2.2 Algorithm for the verification single template

```
genuine_claims = 0
impostor_claims = 0
GA = GR = FA = FR = 0
for each row i: #for each probe
    ordered_similarities <- Descending ordered list of tuples (label_j, similarity)
    for every column j: #for each template
```

```
            if similarity >= threshold: #If the templates are similar enough
                if label_i == label_j: #If the label of the probe and the template are the
                ↪ same
                    GA++
                    genuine_claims++
                else:
                    FA++
                    impostor_claims++
            else: #If the templates are not similar enough
                if label_i == label_j: #If the label of the probe and the template are the
                ↪ same
                    FR++
                    genuine_claims++
                else:
                    GR++
                    impostor_claims++
GAR = GA / genuine_claims
GRR = GR / impostor_claims
FAR = FA / impostor_claims
FRR = FR / genuine_claims
```

Listing 7.2: Verification single template

7.2.3 Algorithm for the verification multiple template

```
genuine_claims = 0
impostor_claims = 0
GA = GR = FA = FR = 0
for each row i: #for every probe
    genuine_claims++
    ordered_similarities <- Descending ordered list of tuples (label_j, similarity)
    best_similarities <- Database with the best match between the current probe and a
    ↪ template from each class
    for each column j: #for each identity
        impostor_claims++
        if best_similarity[j] >= threshold: #If the templates are similar enough
            if label_i == label_j: #If the label of the probe and the template are the same
                GA++
            else:
                FA++
        else: #If the templates are not similar enough
            if label_i == label_j: #If the label of the probe and the template are the same
                FR++
```

```
18              else:
19                  GR++
20 GAR = GA / genuine_claims
21 GRR = GR / impostor_claims
22 FAR = FA / impostor_claims
23 FRR = FR / genuine_claims
```

Listing 7.3: Verification single template

7.3 Evaluation procedure

Let's see how we performed the evaluation for each one of the methods in detail, and let's compare the results.

For each test with our methods, we made the plots for the ROC (Receiving Operating Curve) and FRR-FAR curves.

The ROC curve is used to find the AUC (Area under the ROC) and depicts the probability of Genuine Accept (GAR) of the system, expressed as 1-FRR vs FAR variation as y and x axes. The best possible prediction method would yeild a point in the upper left corner of the ROC space, representing no false negatives and no false positives. The points above the diagonal that divides the ROC space represent good results (better than random), the points below the line represent bad results.

In the plot with FRR-FAR curves we can find the EER (Equal Error Rate), and the ZeroFAR, ZeroFRR points. FAR and FRR both depend on the similarity/distance threshold yet in opposite directions: if we raise the threshold, the detect and identify rate decreases, but our false alarm rate also decreases; if we lower the threshold, the detect and identify rate increases, but our false alarm rate also increases. The EER is the value of the threshold when FAR=FRR and is the value used as the acceptance threshold, the ZeroFRR is the value of FAR when FRR≈0 and the ZeroFAR is the value of FRR when FAR≈0. The best pair of FAR and FRR is the one for which the EER has the lowest y-coordinate

7.3.1 VGGFace

Regarding the VGGFace model, the procedure is the following:

- Load the images from the dataset as `X` and the labels as `y`;
- Replace each image in `X` with the relative feature vector extracted;
- Split the feature vectors list `X` and the labels `y` in training (66%) and testing (33%) set. The feature vectors in the training set will be the gallery, and the ones in the testing set will be the probes;
- Build the similarity matrix that, for simplicity in future use in the all-probes-against-all-gallery algorithm, we decided to implement with the following data structure:

```
[
(id_probe_1,
    [(id_gallery_1, similarity_1_1),
    (id_gallery_2, similarity_1_2),
    ...,
    (id_gallery_n, similarity_1_n)]),
(id_probe_2,
    [(id_gallery_1, similarity_2_1),
    (id_gallery_2, similarity_2_2),
    ...,
    (id_gallery_n, similarity_2_n)]),
...,
(id_probe_m,
    [(id_gallery_1, similarity_m_1),
    (id_gallery_2, simiarity_m_1),
    ...,
    (id_gallery_n, similarity_m_n)])
]
```

That is a list of tuples where the first element is the id of the probe feature vector $i \in [1, m]$, and the second element is another inner list of tuples where the first element is the id of gallery feature vector $j \in [1, n]$ and the second element is the similarity between feature vector i and feature vector j.

- Run the all-probes-against-all-gallery algorithm for the open set identification, described in subsubsection 7.2.1 with the computed similarities and for all thresholds in the $[0, 1]$ interval with a 0.01 increment in order to get the FAR, FRR, GAR, GRR, DIR metrics for all the possible thresholds.

We start by visualizing the performances of VGGFace by only taking people in the LFW dataset that have at least 4 faces of their faces, since in our system that's the minimum quantity of photos that are required to be enrolled.

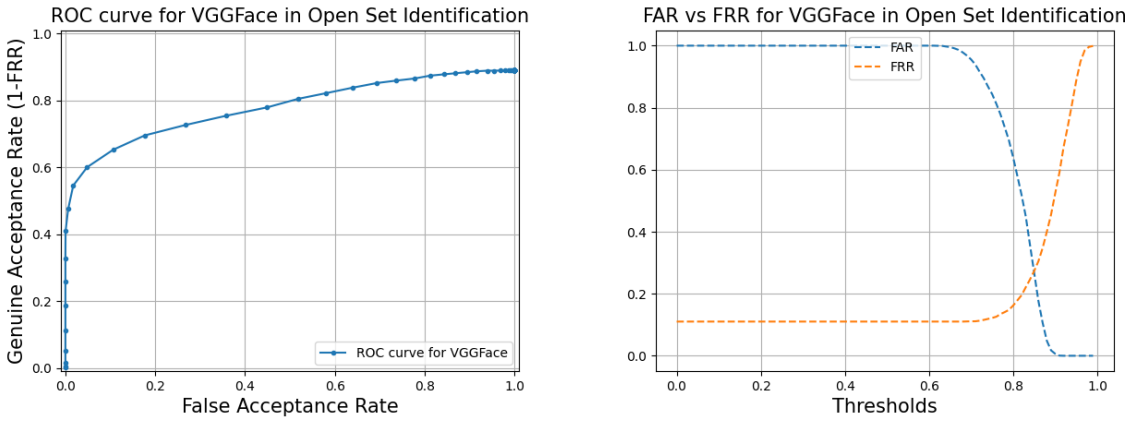

Figure 15: Open set identification for LFW Dataset with 4 faces per person
AUC = 0.78, EER = 0.27 with threshold = 0.85
ZeroFAR = 0.67, ZeroFRR = Not Defined,

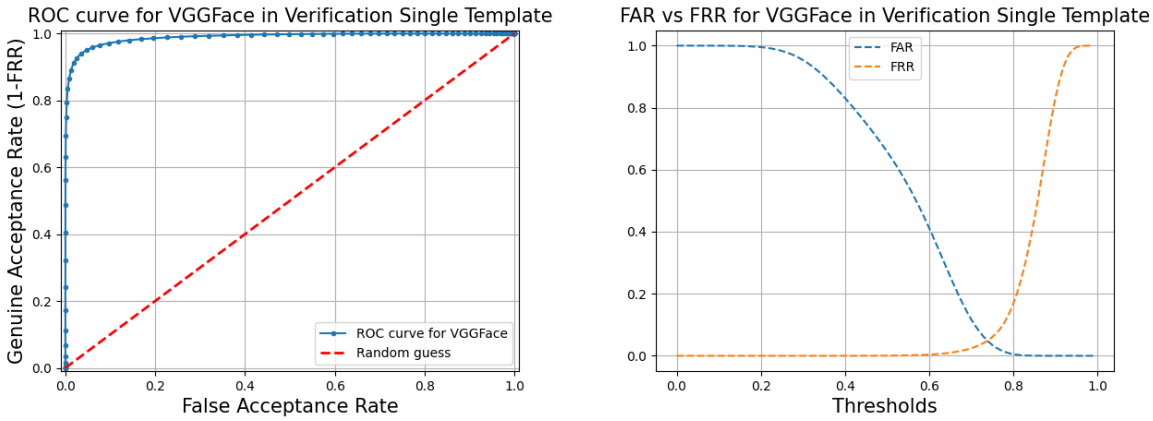

Figure 16: Verification single-template for LFW Dataset with 4 faces per person
AUC = 0.989, EER = 0.05 with threshold = 0.74
ZeroFAR = 0.59, ZeroFRR = 0.77,

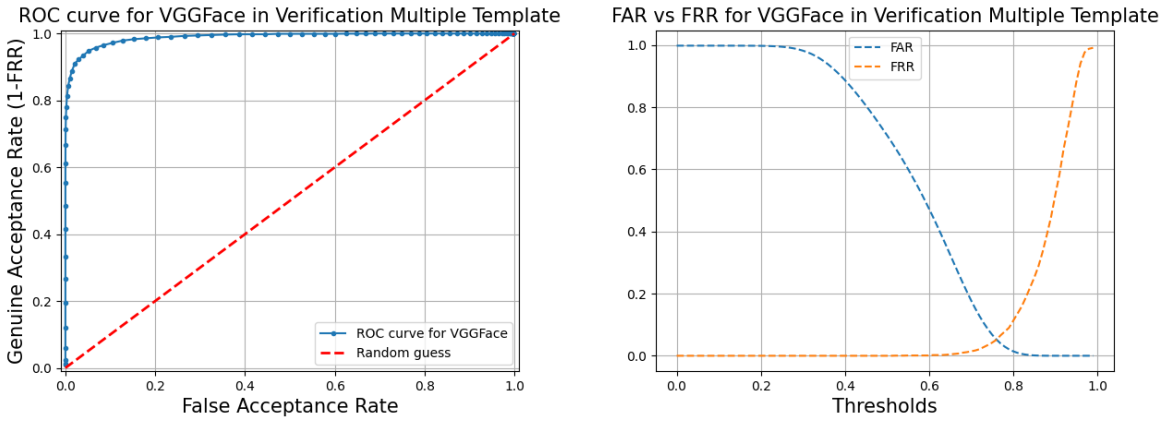

Figure 17: Verification multiple-template for LFW Dataset with 4 faces per person
AUC = 0.987, EER = 0.05 with threshold = 0.76
ZeroFAR = 0.45, ZeroFRR = 0.69,

In the plots we can observe that for the Open Set Identification we have a good ROC curve but we can't define ZeroFRR because FRR does not reach the value of zero. In the verification, the results are better with an excellent ROC curve and a lowest EER with lowest threshold.

We can see how the model performs better with more info about each user if we only take people from the dataset that have at least 7 pictures of their faces:

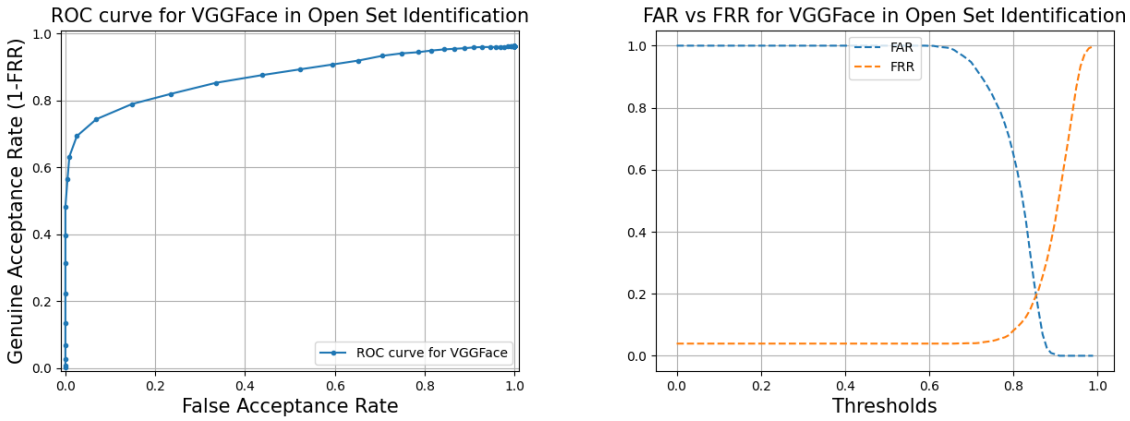

Figure 18: Open set identification for LFW Dataset with 7 faces per person
AUC = 0.87, EER = 0.207 with threshold = 0.85
ZeroFAR = 0.52, ZeroFRR = Not Defined,

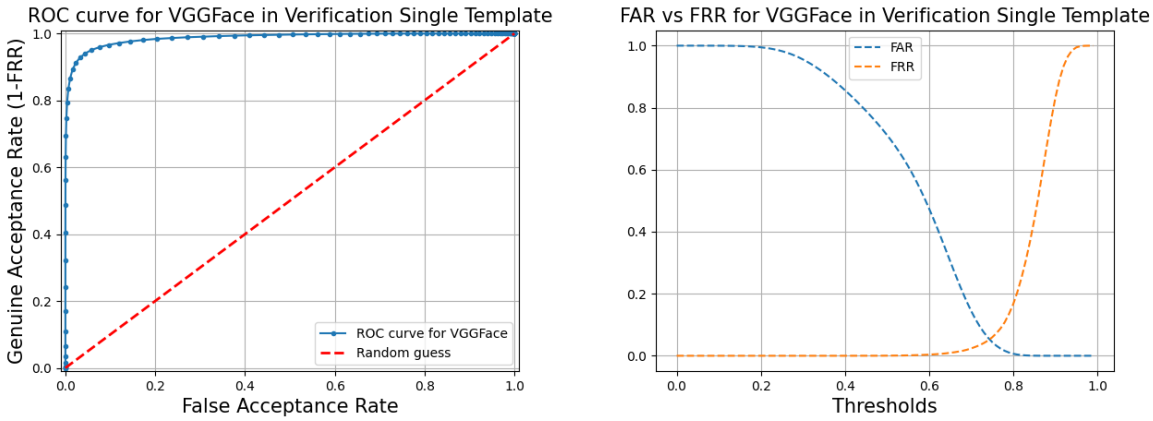

Figure 19: Verification single-template for LFW Dataset with 7 faces per person
AUC = 0.987, EER = 0.053 with threshold = 0.74
ZeroFAR = 0.59, ZeroFRR = 0.81,

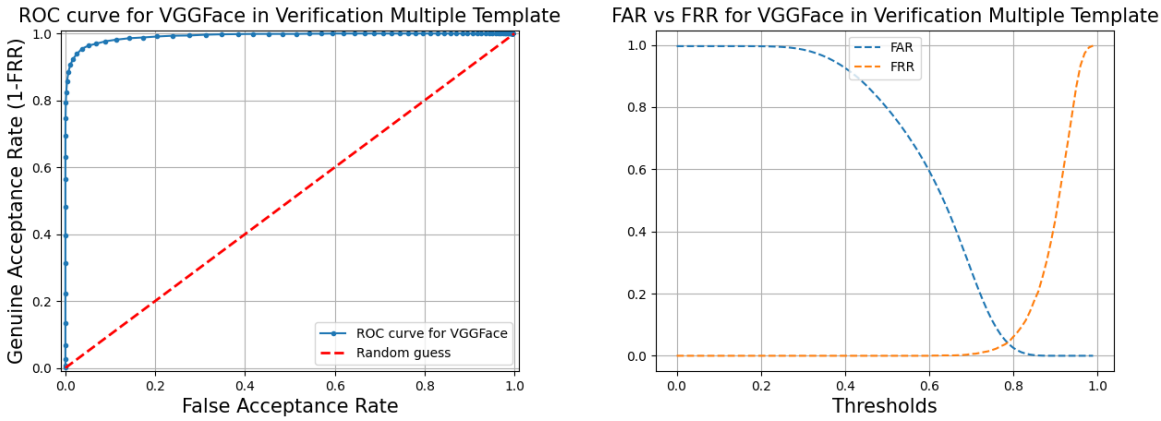

Figure 20: Verification multiple-template for LFW Dataset with 7 faces per person
AUC = 0.988, EER = 0.04 with threshold = 0.79
ZeroFAR = 0.37, ZeroFRR = 0.59,

We can observe that with at least 7 pictures ROC curve is better in the Open Set Identification and also the value of AUC is bigger. The EER and the threshold are the same as the results with 4 pictures.

Since Labeled Faces in the Wild is a dataset containing faces with exaggerated variation in pose, illumination, and expression, we also tested the model on the Olivetti faces dataset, which faces are more suited to our system's use-case, since they're more uniform in pose, expression, and illumination. We can see that performances improve a lot.

Figure 21: Open set identification for Olivetti faces dataset
AUC = 0.985, EER = 0.06 with threshold = 0.84
ZeroFAR = 0.14, ZeroFRR = 0.65,

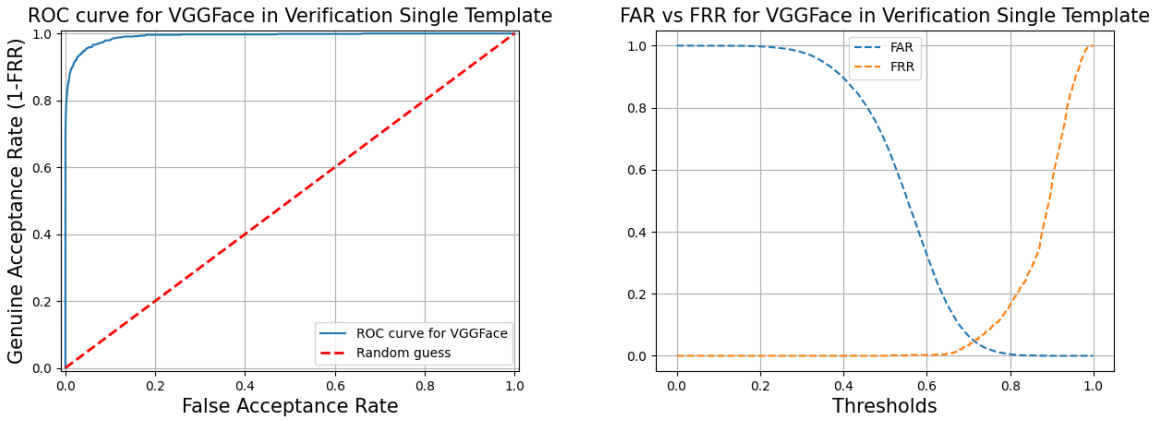

Figure 22: Verification single-template Olivetti faces dataset
AUC = 0.991, EER = 0.046 with threshold = 0.715
ZeroFAR = 0.33, ZeroFRR = 0.66,

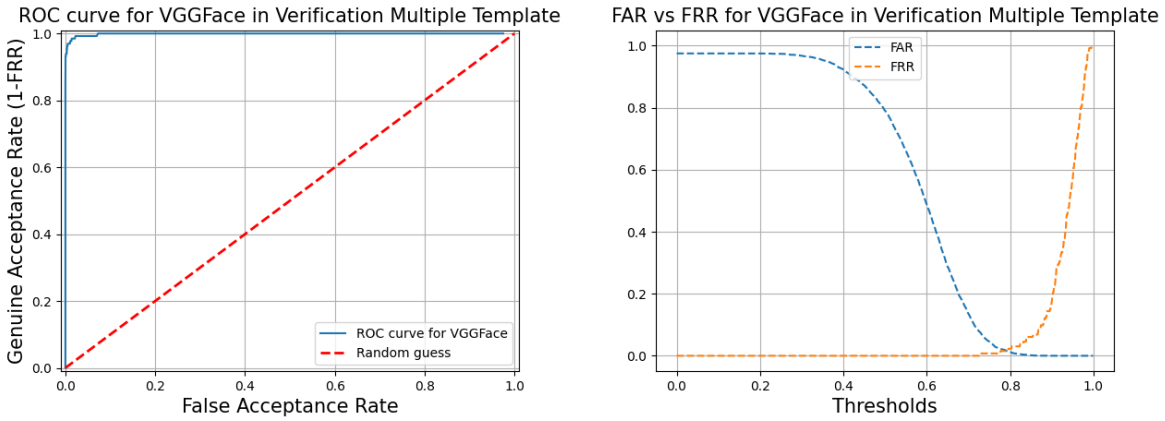

Figure 23: Verification multiple-template for Olivetti faces dataset
AUC = 0.973, EER = 0.015 with threshold = 0.79
ZeroFAR = 0.14, ZeroFRR = 0.07,

We can observe that the results are better with the Olivetti dataset, especially in the Open Set Identification where the ROC curve is excellent and the value of AUC is bigger. Also, the EER is lower but the threshold has the same value.

7.3.2 LBPH

LBPH has almost the same procedure as the VGGFace model:

- After loading the images from the dataset as `X` and the labels as `y` we will replace each image in `X` with the relative feature vector extracted which in this case will correspond to the normalized histogram of the image;
- Then split the feature vectors list `X` and the labels `y` in training and testing set, before building the similarity matrix with the same structure described previously for VGGFace;
- In the end, run the all-probes-against-all-gallery algorithm for the open set identification, with the computed correlation and for all thresholds to get the FAR, FRR, GAR, GRR, DIR metrics for all of them;

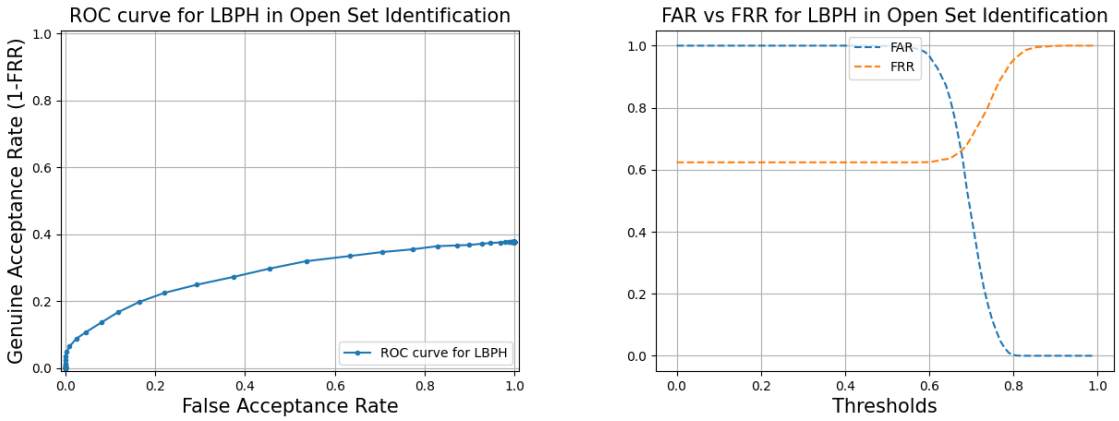

Figure 24: Open set identification for LFW Dataset with 4 faces per person
AUC = 0.28, EER = 0.65 with threshold = 0.68
ZeroFAR = 0.98, ZeroFRR = Not Defined,

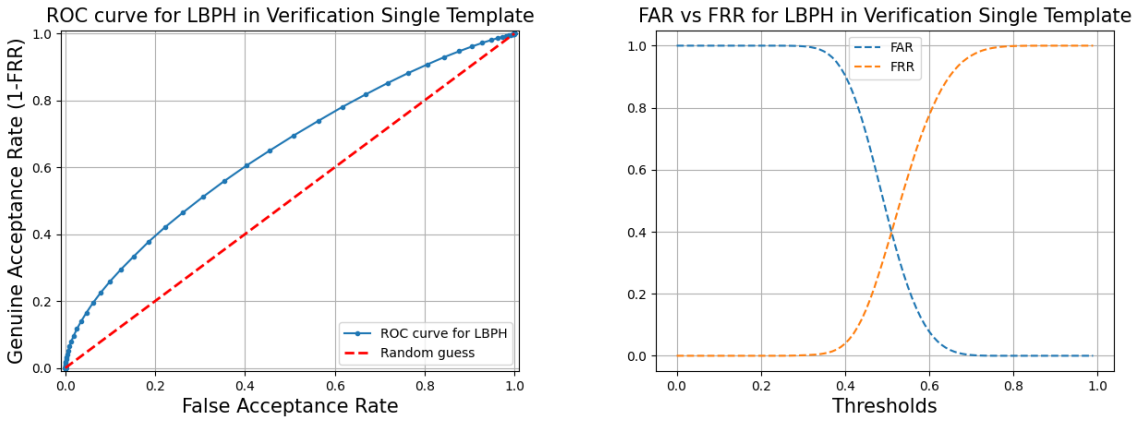

Figure 25: Verification single-template for LFW Dataset with 4 faces per person
AUC = 0.646, EER = 0.4 with threshold = 0.51
ZeroFAR = 0.994, ZeroFRR = 0.999,

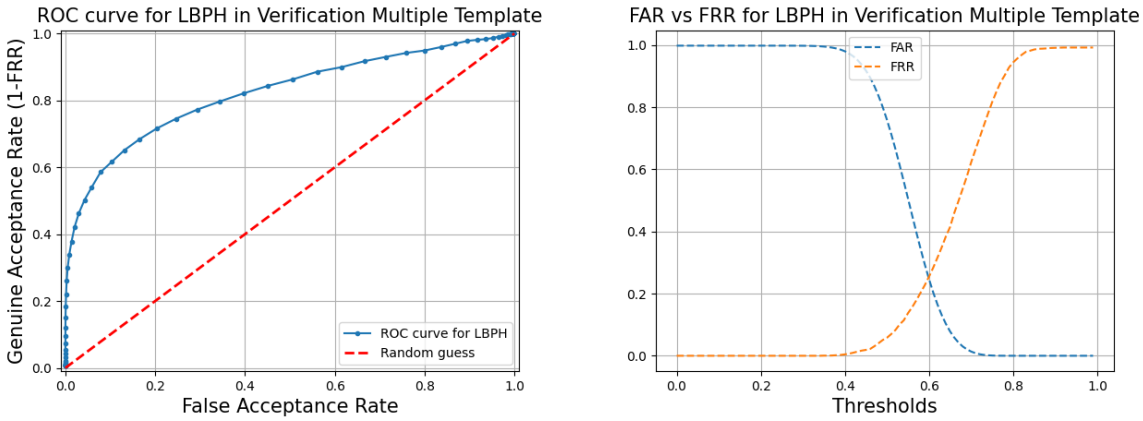

Figure 26: Verification multiple-template for LFW Dataset with 4 faces per person
AUC = 0.82, EER = 0.25 with threshold = 0.60
ZeroFAR = 0.90, ZeroFRR = 0.996,

In the plots, we can observe that in the Open Set Identification the ROC curve has points below the diagonal. This means that the algorithm doesn't perform so well with the LFW Dataset in the case of a minimum of 4 faces per person. In the verification, the results are a little bit better, especially in the case of multi-template verification, but still not sufficient.

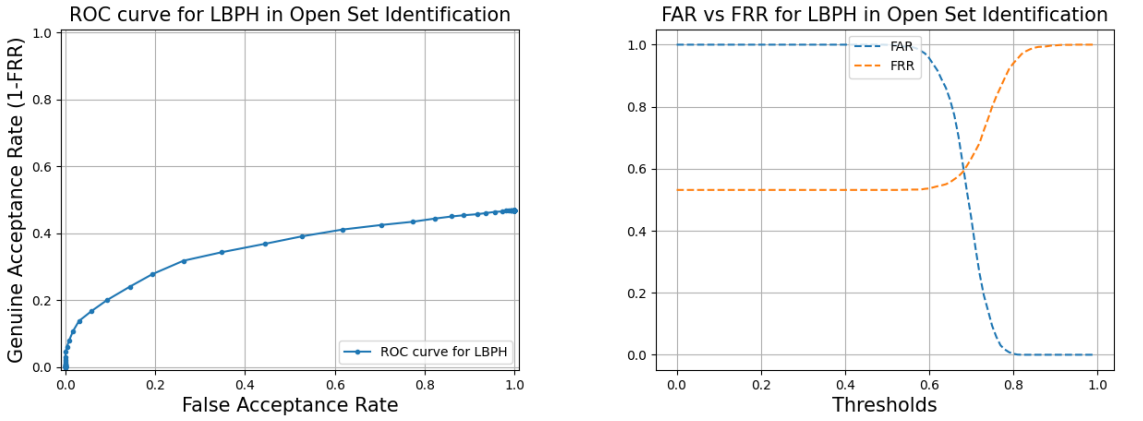

Figure 27: Open set identification for LFW Dataset with 7 faces per person
AUC = 0.35, EER = 0.60 with threshold = 0.68
ZeroFAR = 0.97, ZeroFRR = Not Defined,

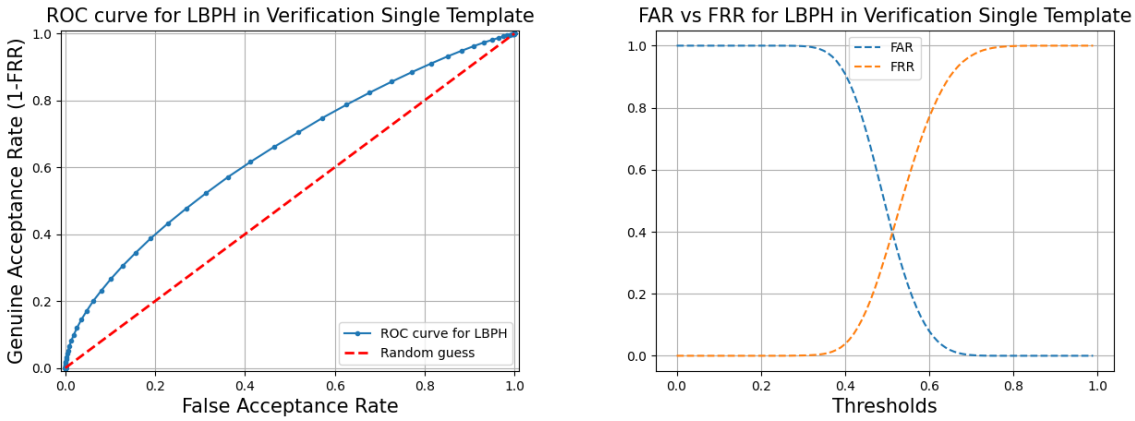

Figure 28: Verification single-template for LFW Dataset with 7 faces per person
AUC = 0.647, EER = 0.398 with threshold = 0.51
ZeroFAR = 0.994, ZeroFRR = 0.999,

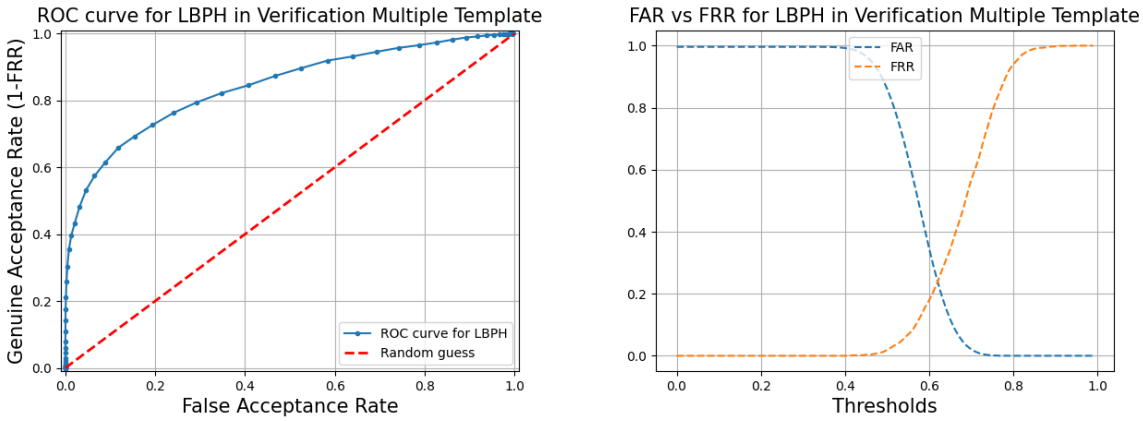

Figure 29: Verification multiple-template for LFW Dataset with 7 faces per person
AUC = 0.84, EER = 0.24 with threshold = 0.62
ZeroFAR = 0.90, ZeroFRR = 0.992,

As seen for the previous model, increasing the minimum number of faces per person in the training and testing set gives us better results, but they're still not sufficient for reliable identification. In the verification, the results are very similar to the previous example with 4 faces.

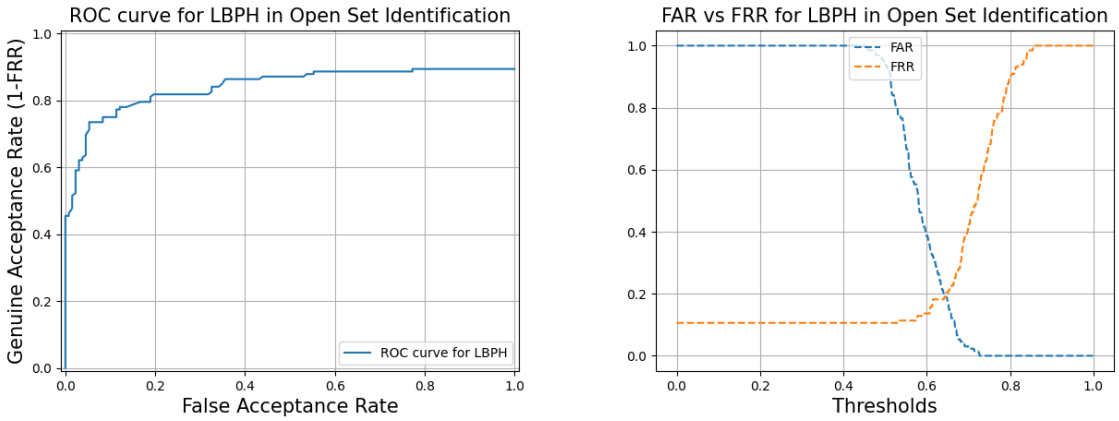

Figure 30: Open set identification for Olivetti faces dataset
AUC = 0.84, EER = 0.19 with threshold = 0.64
ZeroFAR = 0.54, ZeroFRR = Not Defined,

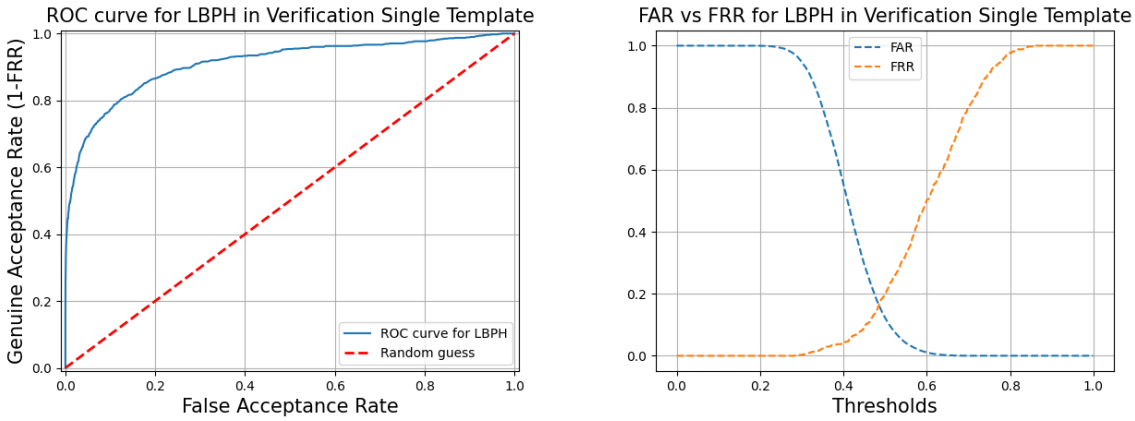

Figure 31: Verification single-template Olivetti faces dataset
AUC = 0.911, EER = 0.16 with threshold = 0.485
ZeroFAR = 0.83, ZeroFRR = 0.97,

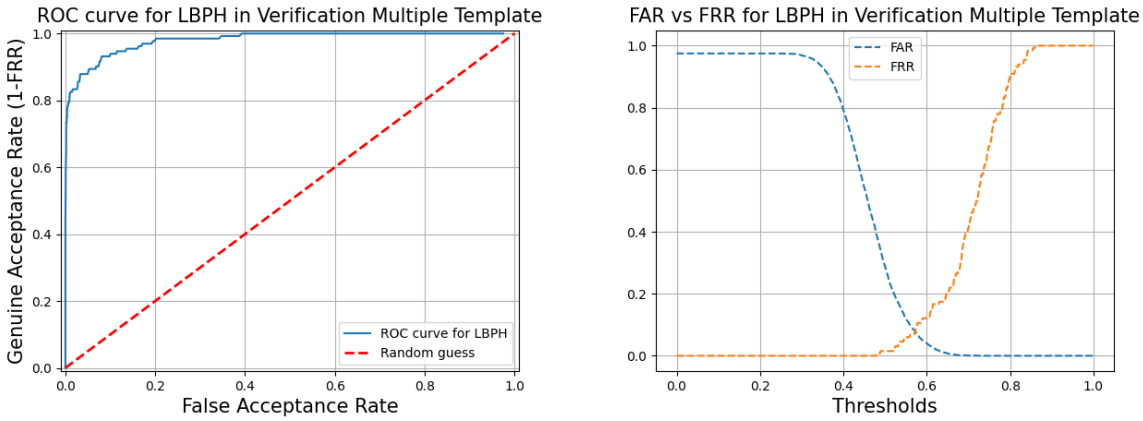

Figure 32: Verification multiple-template for Olivetti faces dataset
AUC = 0.955, EER = 0.077 with threshold = 0.57
ZeroFAR = 0.54, ZeroFRR = 0.39,

In these plots, we can observe the differences with the Olivetti dataset. We have good results in Open Set Identification, due to the simpler dataset, with the ROC curve that has most of the points in the upper part of the plot. Also in the Verification, the results are better with a lower EER.

7.3.3 SVC

Since the SVC couldn't be used as a feature extractor, and it was used directly to predict the probability that a certain template has to be of a certain identity, the procedure is a little bit different than the other two methods:

- Load the images from the dataset as `X` and the labels as `y`;
- Convert each image in the `X` set to a 128-dimensional array that represents its histograms of gradients.
- Split the histograms of gradients list `X` and the labels `y` in training (66%) and testing (33%) set.
- Train the SVC with the training set. In this case, the SVC itself would be the gallery, since if there aren't any templates of a certain identity i in the training set, then the SVC would never be able to recognize a probe of identity i, and so it's like the user is not in the gallery;
- Build the similarity matrix, calling the `predict_proba(X_test)` method of Sklearn's SVC, that returns a matrix of shape (n, k) where n is the cardinality of the testing set and k is the number of classes (different identities in the training set). In the (i, j) coordinates there is the probability for the template i to be of identity j;
- In order to interpret the probabilities as similarities, we have to overcome the problem that the probability vector sums to 1. This is a problem because if the probe has a probability of 0.6 to

be part of the identity i, then there will be no other probability greater than 0.4, and so the computation of metrics such as DIR could be incorrect. Since there are many classes, this implies that most of the probabilities could be very small even if the actual similarity between the probe class and the predicted class isn't so small. In order to solve this problem, we apply the inverse softmax function:

$$f(x) = \log x + \log \sum_{i}^{k} e^{x_k} \quad \text{with k number of classes} \tag{12}$$

to stretch the numbers from a range of [0,1] to a range of $[-\infty, +\infty]$. Then the array elements are again reduced to a range of [0,1] by applying the sigmoid function:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{13}$$

to each element. This assures us that the order of the elements isn't changed, and so we can interpret the values as a similarity measure since the new values are still proportional to the values before the transformation.

We can see an example of the transformation:
This is the original vector of probability, where each element is a tuple of the form `(class, probability)`:

```
[(29, 0.189), (14, 0.080), (17, 0.0549), (0, 0.053), (15, 0.039), ...]
```

Then we apply the inverse softmax to each of the elements:

```
[(29, 2.048), (14, 1.199), (17, 0.813), (0, 0.792), (15, 0.483), ...]
```

And then we apply the sigmoid:

```
[(29, 0.885), (14, 0.768), (17, 0.692), (0, 0.688), (15, 0.618), ...]
```

As we can see the order of the classes that refers to the probability (the first element of the tuple) never changes.

- Parse the previously computed matrix of probabilities into the same shape as seen for the similarity matrix in the VGGFace procedure, in order to be able to use the same implementation of the algorithm.

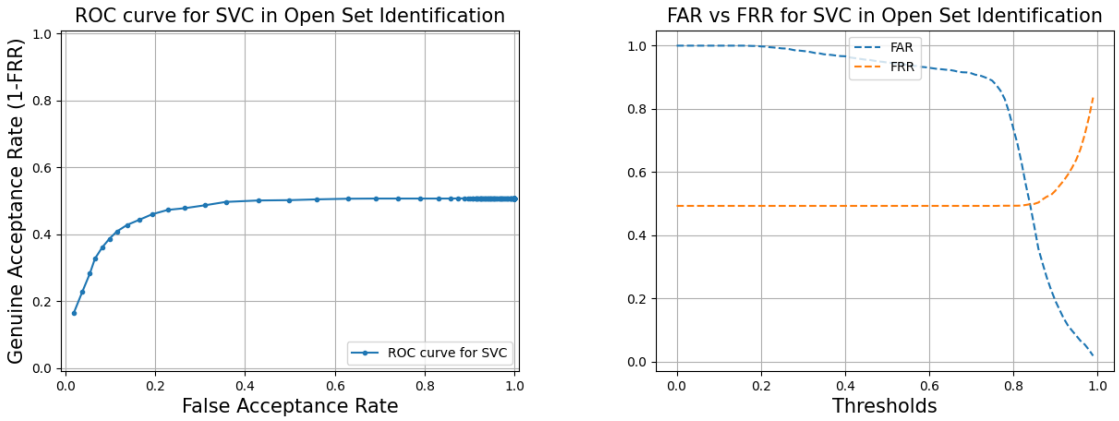

Figure 33: Open set identification for LFW Dataset with 4 faces per person
AUC = 0.47, EER = 0.50 with threshold = 0.84
ZeroFAR = Not Defined, ZeroFRR = Not Defined,

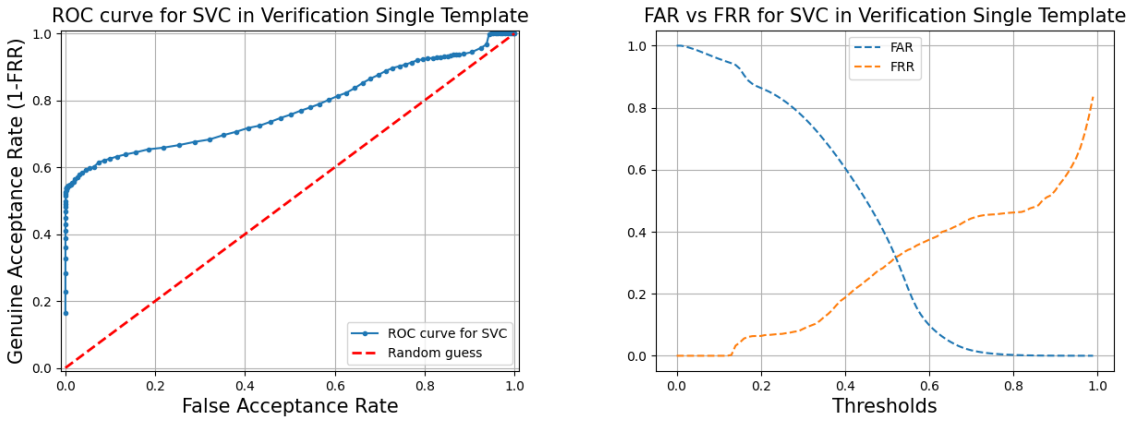

Figure 34: Verification single-template for LFW Dataset with 4 faces per person
AUC = 0.776, EER = 0.32 with threshold = 0.52
ZeroFAR = Not Defined, ZeroFRR = 0.95,

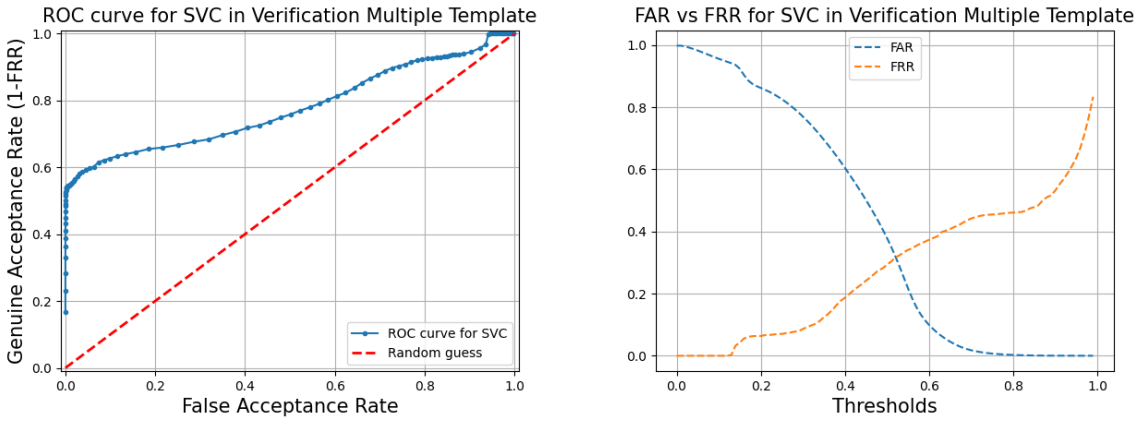

Figure 35: Verification multiple-template for LFW Dataset with 4 faces per person
AUC = 0.776, EER = 0.32 with threshold = 0.52
ZeroFAR = Not Defined, ZeroFRR = 0.95,

As seen before, most of the models have not-so-good performances with LFW with 4 faces per person, and this model isn't an exception. In the case of Open set Identification most of the points in the ROC are in the lower part of the plot. The results are a bit better if we consider the verification case.

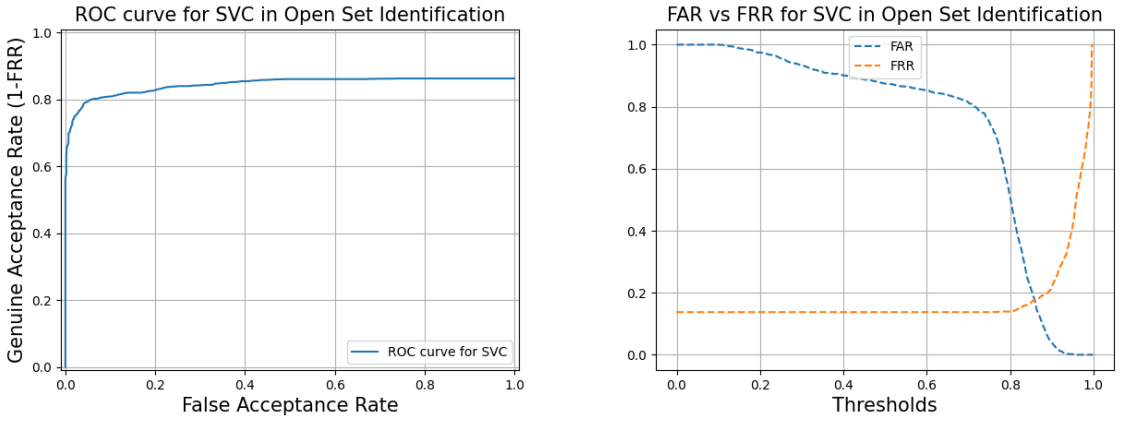

Figure 36: Open set identification for LFW Dataset with 7 faces per person
AUC = 0.84, EER = 0.178 with threshold = 0.86
ZeroFAR = 0.44, ZeroFRR = Not Defined,

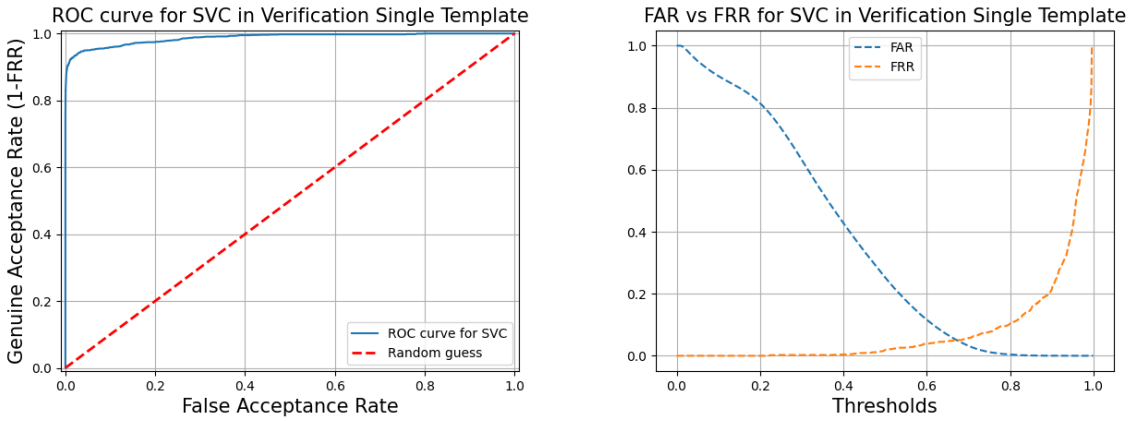

Figure 37: Verification single-template for LFW Dataset with 7 faces per person
AUC = 0.987, EER = 0.05 with threshold = 0.67
ZeroFAR = 0.24, ZeroFRR = 0.79,

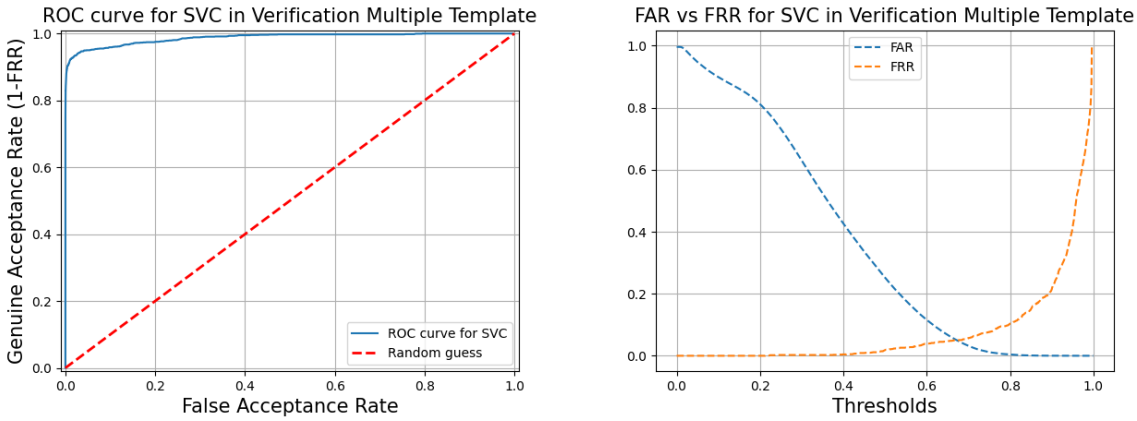

Figure 38: Verification multiple-template for LFW Dataset with 7 faces per person
AUC = 0.983, EER = 0.05 with threshold = 0.67
ZeroFAR = 0.24, ZeroFRR = 0.79,

With 7 faces per person, the results are better in Open Set Identification but we still have points of the ROC curve below the diagonal. In the verification, the ROC curve is excellent and the EER is also lower.

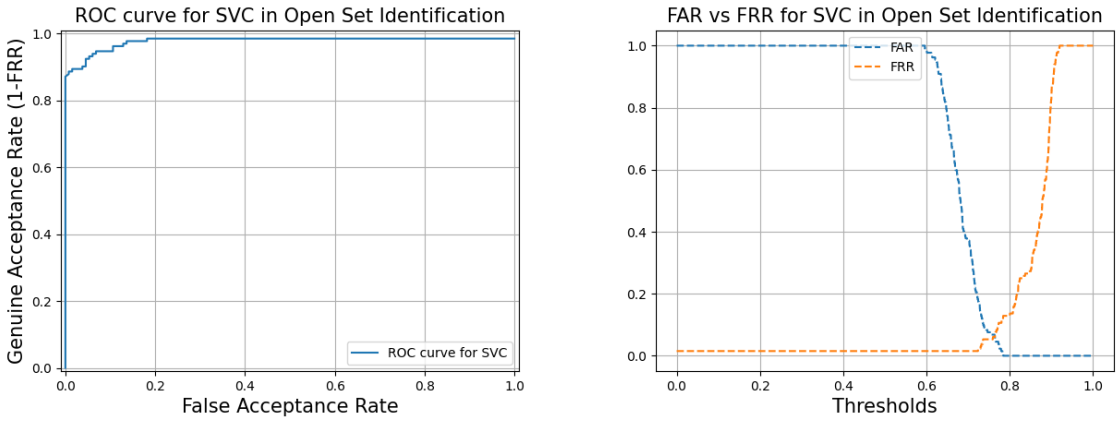

Figure 39: Open set identification for Olivetti faces dataset
AUC = 0.976, EER = 0.06 with threshold = 0.76
ZeroFAR = 0.13, ZeroFRR = Not Defined,

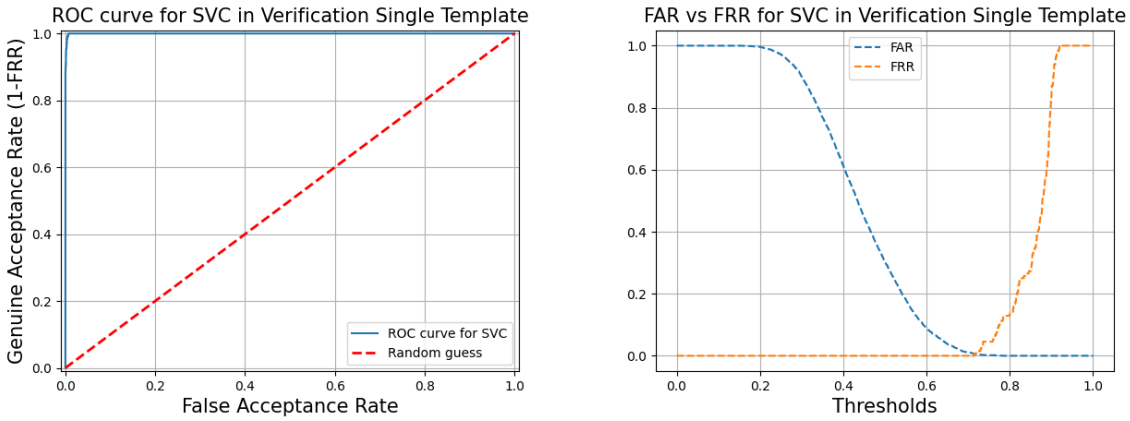

Figure 40: Verification single-template Olivetti faces dataset
AUC = 0.999, EER = 0.007 with threshold = 0.714
ZeroFAR = 0.13, ZeroFRR = 0.006,

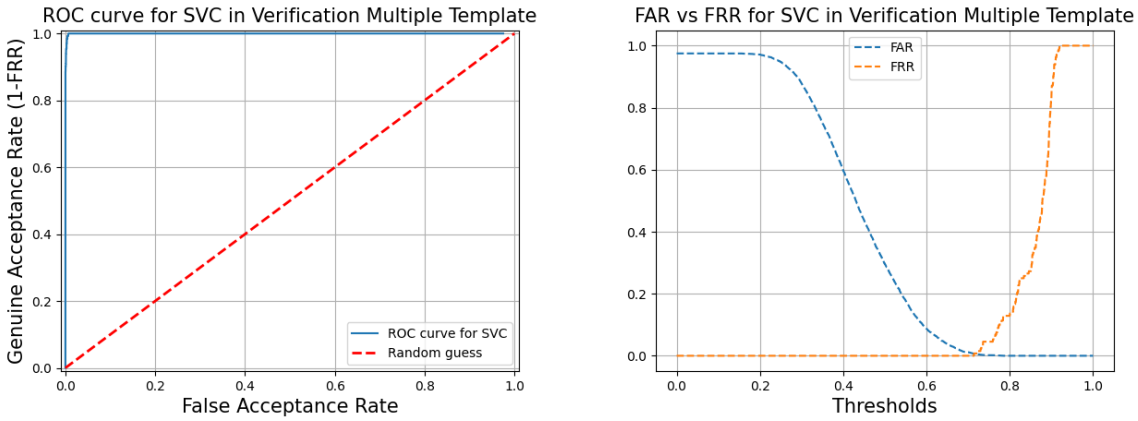

Figure 41: Verification multiple-template for Olivetti faces dataset
AUC = 0.975, EER = 0.007 with threshold = 0.71
ZeroFAR = 0.13, ZeroFRR = 0.006,

With the Olivetti dataset, the ROC curve is excellent also in the case of Open Set Identification. We also obtain a reasonable value for the EER in all of the cases.

7.4 Evaluation conclusions

The results with the Olivetti dataset are better than the results with the LFW dataset, so we can use the first one to make some comparisons and choose the best model. In the Open Set Identification, we can observe that the VGG model is the best because it has the bigger AUC=0.985 and the smallest EER=0.06. The SVC model has very similar results with the AUC=0.976 and the EER=0.06. The LBPH has the lower results with AUC=0.84 and EER=0.19 but they still are good results. VGG performs better with a high threshold that is equal to 0.85. On the contrary, we have LBPH that performs better with low thereshold=0.64. In the middle, we have SVC with thereshold=0.76.

Also in the Verification single template and multi template, the VGG model and the SVC model are the best with similar results. We can also observe that for both models, the results in the Verification are better than the results in Identification Open Set.

With the LFW dataset, we can observe that the results are good for VGG model in both cases, with 4 faces and 7 faces. On the contrary, LBPH has bad results in both cases. SVC has bad results with 4 faces but has good results with 7 faces.

8 Experiments

In this section we have collected all the experiments that were carried out during the implementation phase which were then discarded with the relative motivation.

8.1 Application of filters

We had decided to use filters to improve system performance, their application consisted in generating mirror images for each image that was uploaded through the enrollment phase by the user with changes on parameters such as contrast, saturation, grayscale, and brightness. In fact, there have been some small improvements in the recognition phase, thanks to the greater number of photos for each user which allowed us to acquire more details based on the applied filter. We also found the same improvements during the evaluation phase, but unfortunately the approach greatly weighed down the system due to the large number of images that had to be processed, this translated into a significant lengthening of image processing times. For this reason, since the improvement was minimal, we decided not to use them in the final system.

8.2 Identification of the three bands (eyes, nose and mouth)

Instead of using only the Haar Cascade classifier that detects only the face we tried to recreate it from the combination of eyes, nose and mouth detected thanks to the dedicated Haar Cascades classifiers. Unfortunately, they did not give us satisfactory results because the characteristics extracted from their combination were not sufficient to make a comparison, returning results clearly lower than the one dedicated to the entire face.

8.3 Face alignment

To see if it was possible to further improve face recognition we tried to implement a face aligner. Its operation was very simple, the image of the person is rotated according to the angle of the eyes. However, given that in the case of enrollment and recognition we are in a controlled environment, we have not noticed substantial differences regarding face recognition. Same thing for the evaluation phase where we used datasets already optimized for this purpose.

9 Conclusions

The implementation of this project allowed us to find a possible effective solution for solving some of the problems that occur every day at the university cafeteria: the long queue that is created to access it and the QR Code system that besides slowing down the queue can be easily spoofed. In order to solve these problems, we designed and implemented a dedicated user interface for both the administrator and the students to best simplify the experience and, most importantly, to speed up student recognition through the use of face recognition. This would greatly help to reduce waiting time as recognition becomes instantaneous since it is done in a situation that is controlled and supervised by an operator and difficult to attack since students must physically present themselves to be recognized in order to gain access. Speaking more specifically, we saw and compared 3 different models: LBPH, VGG Face and SVC. According to the results obtained from the evaluation phase, we saw how our system, in general, performs better when the environment is controlled and the user is perfectly centered with the camera. Among the various models tested we find in first place VGG Face, which performed very well in both Open Set: Identification and Verification: single and multiple templates even in situations where the user was not perfectly centered. In second place we find SVC, which also had very good results but slightly worse when the user was not in the correct position. In last place we find LBPH which returned results that were not very satisfactory compared to the other two. Therefore, we decided to use VGG Face as the definitive method for our system.

References

[1] Local Binary Patterns Histograms https://towardsdatascience.com/face-recognition-how-lbph-works-90ec258c3d6b

[2] Pearson's correlation coefficient https://en.wikipedia.org/wiki/Pearson_correlation_coefficient

[3] Support Vector Machine https://en.wikipedia.org/wiki/Support_vector_machine

[4] VGG Face Descriptor https://www.robots.ox.ac.uk/~vgg/software/vgg_face/

[5] Labeled Faces in the Wild http://vis-www.cs.umass.edu/lfw/

[6] Scikit Learn - The Olivetti faces dataset https://scikit-learn.org/stable/datasets/real_world.html#olivetti-faces-dataset