Biometric Wallet
Biometric Systems group project

Davide Basile 1810355
Niccolò Morabito 1808746
Valerio Goretti 1811110
Luca Pierfederici 1754894

April 2021

Contents

Chapter 1

Introduction

1.1 Biometric Wallet System

A wallet, in the blockchain world, is a system that stores important information about the blockchain account of its users (seeds, balances, public key and private key). This information is essential to sign and send transactions to the blockchain's network. The main idea of this project is to use biometric traits in order to create a service of authentication that defends critical information of the blockchain accounts. The two main biometric operations used for the authentication are:

- 2D Facial Verification
- Voice Verification

1.2 Document Overview

In this section, it is possible to find an overview of the structure of the rest of the document. Chapter 2 describes the design choices, the technologies and the architecture of the system. Chapter 3 explains 2D facial verification and its two main components: enrollment and verification. Moreover, it is possible to find the general approach and methodology used for the 2D facial verification. Chapter 4 is focused on voice recognition’s operations. In the end, in chapter 5, the evaluation technique and results are explained.

Chapter 2

Use case, architecture and technologies

2.1 The blockchain: IOTA devnet

As said before, a wallet is used to store sensitive information about a blockchain account, that is necessary to send and receive values. The blockchain to which the system refers is IOTA devnet. IOTA is a new DLT technology, designed to support Internet of Things operations and to store resulting microtransactions. There are different IOTA networks, and each of them has its own blockchain ledger. For this case, the network used is the devnet, intended to be used by developers.

2.2 Use case

In the application there are the following use cases:

- Enroll your wallet
  - Record your face
  - Record your voice
  - Generate your account

- Send Currency: transfer an amount of currency to another wallet
  - Insert the amount of currency that you would sent
  - Insert the destination address
  - Verify your voice to unlock the transaction

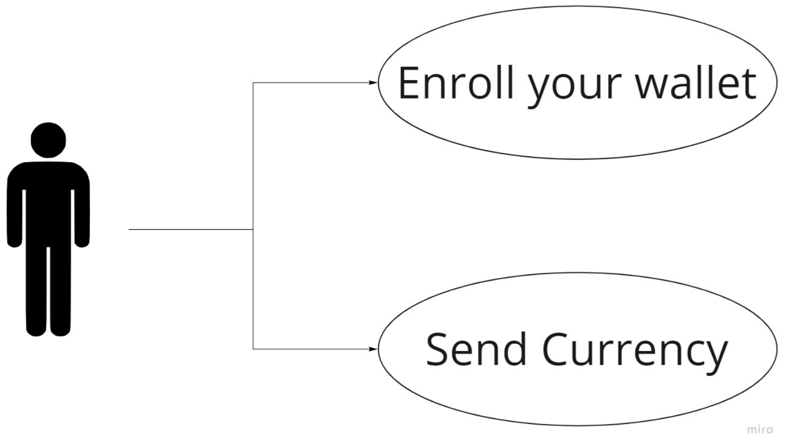


Figure 2.1: Use Case model

2.3 Use case descriptions

2.3.1 Enrollment

The app allows to create and store a IOTA wallet on a mobile device. The user opens the app and can register his wallet. During the registration the user provides his credentials (username and password), then he uses the 2D facial tool to store his facial features in the application. More information on how 2D facial recognition works can be found in the next chapter.

After the 2D face enrollment, the user will record audio up to 20 seconds that will be used to create his vocal profile. The created profile is his "fingerprint" that will be used to verify every transaction of him. After these steps, the user will see a summary of his registration data and clicking on the confirm button he will be redirected to the initial page where he can login and make a transaction from.

2.3.2 Login and transactions

After the login, the user may have the need to send his money to another wallet. In order to do this, the user must enter the amount of currency that would like to transfer and the address he wants to send them to. After entering this data the user records a phrase to ensure that he is the correct person who makes the transaction.

The verification is done if the voice recorded has a score of similarity that ensures the person is the correct one. When the verification is done the selected amount of money is sent to the specified address.

2.4 System Architecture

The architecture of the system (Fig. 2.2) is based on the Android platform and on some external services for voice verification and IOTA system that the Android application interfaces with. In particular, in addition to the Android app we use:

- Server API for the IOTA Service
- Microsoft Azure API for Voice Verification

The IOTA service is used for connecting with the IOTA network. In order to do this, we need to connect to a node in the IOTA network and when the user logs in or sends a transaction, we must connect with his wallet to the node and then to the network.

When we create a IOTA Account for the wallet, we need a username and a password. To make sure that the user does not have to remember it, we use a hash of the features of facial recognition as a password. The Microsoft Azure API is the second external service, essential to verify if the user is really who he claims to be. In fact, it guarantees that the person who uses the wallet, once passed facial recognition, also has the exact voice to unlock the transaction.

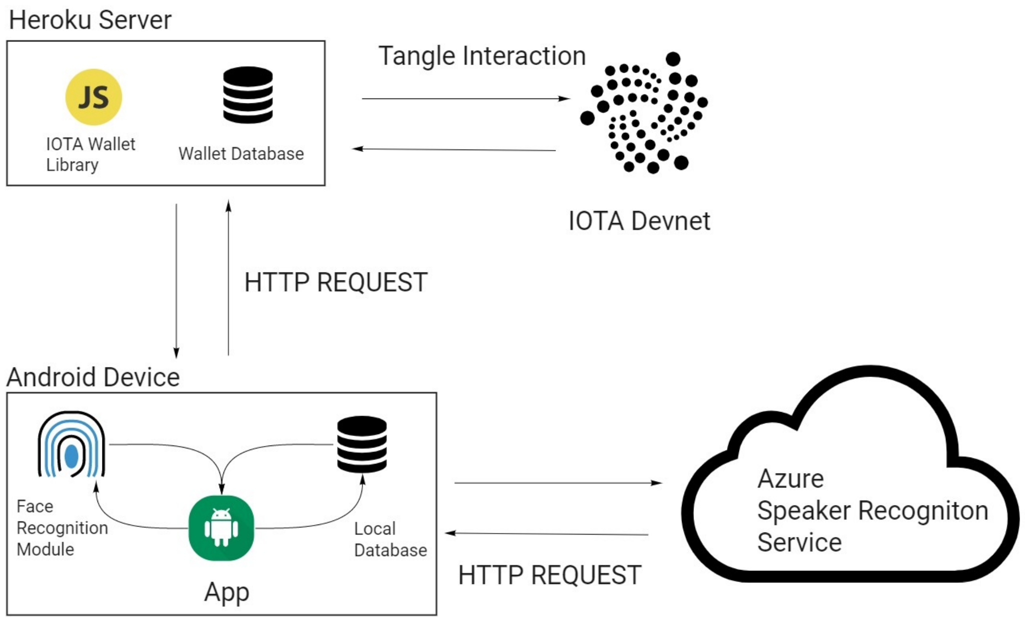


Figure 2.2: Architecture of the system

2.5 Biometric traits

The biometric traits chosen for the recognition tasks are face and voice. The face identification is used to perform two different actions:

- Login
- Data Encryption

The login operation it's done to get access to the account information (balances, seed, addresses etc.). All these data are stored in an online server, and accessed through HTTP requests. In order to protect this sensitive information, a mechanism called 'Stronghold' is used to encrypt the files on the server. This algorithm uses a given key to perform the operation of encryption. Every time the user wants to access the protected data, he needs to provide the key, to perform the decryption operation. In our system, it has been chosen to use, for every account, the hash of the face template obtained during the enrollment process. In this way, sensitive data are protected by using features extracted from the user's face.

The main purpose of voice verification is to send money and make transactions. Every time the logged user wants to generate a new transaction, he must provide his voice and, if the result of the verification is positive, the transaction is sent to the blockchain network.

Chapter 3

2D Facial Verification

3.1 General Approach

The operations performed for the 2D Facial Verification are executed locally by the Android device. The two main phases of facial verification are the enrollment and the recognition steps. The enrollment of a new face is performed during the registration task, when a new user associates a face picture with his name. The picture is elaborated by the system and the result of the elaboration is a vectorial representation, stored into the device. When the registration phase is over, the user is enrolled and his name is now linked to the result of the enrollment phase. The recognition phase takes place during the login task.

After the user proposes a username and the relative password, the recognition step is used to verify if the user is who claims to be. Therefore, a new picture is taken and elaborated by the system. The result of this elaboration is compared with the vectorial representation memorized into the system, belonging to the claimed identity. If the relative distance is below a fixed threshold, then the verification returns a positive result.

3.2 Models Used

The verification system that performs the recognition and the evaluation steps relies on machine learning techniques. The two models used in both the steps are the Face Detection module of the ML Kit for mobile devices by Google and the "FaceNet" neural network.

The ML kit by Google offers a set of pre-trained machine learning models, designed to be executed on devices with reduced memory and low computational power, such as smartphones. This model is used to detect faces in a given picture and to retrieve important information about the detected face (the pose, the probability of smile, the probability of open eyes, etc.). In order to increase the accuracy of the whole procedure, since this model can potentially detect more faces in a given picture, it has been decided to consider only the biggest detection result from the model (if there have been detections). In this way, there are higher probabilities that just the user's face is processed (which is in the foreground), and any faces in the background will not be taken into account for post-detection steps.

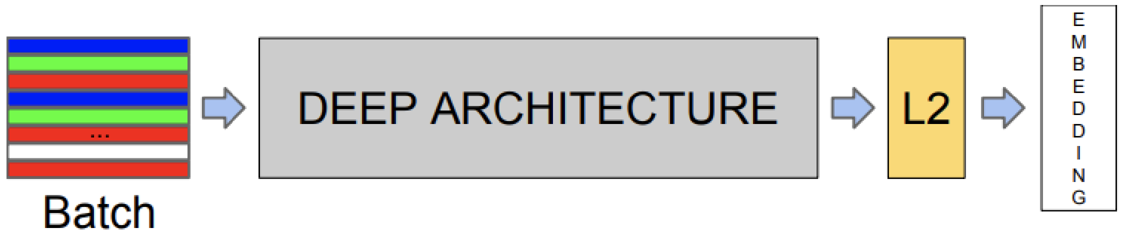


Figure 3.1: FaceNet model architecture

The latter model used is the FaceNet convolutional neural network, one of the most famous models in the field of the face recognition. It is a neural network formed by 22 convolutional layers. This network requires an input image of size 112 x 112 px. Every layer takes as input the result of the previous layer, and applies a convolutional filter (typically formed by a 2d convolution, activation function, and pooling operations) highlighting at each step certain characteristic elements of the face. An important feature of the network is that the operations into the convolutional layers are not performed sequentially like in other CNNs, but by using the inception module (in a parallel way). At the end of the network, the L2 normalization is performed, producing the output as a 128-dimensions vector from the input image. Since the model used for the system is pre-trained, no training phase was required. Moreover, since the whole system is based on the Android execution environment, it was necessary to convert the pre-trained model into a lite version, efficiently runnable by a mobile device. In order to do that, TensorFlow lite library has been used. The file of the model is named "mobile_face_net.tfmobile" and can be found in the "asset" folder of the android project.

3.3 Enrollment

In order to obtain and manipulate the images of the user, the OpenCv library for Android devices has been used. By using the OpenCv camera object, it is possible to process independently frames belonging to a continuous flow of images captured by the device. The number of frames processed per second depends on the complexity of the operations that are executed.

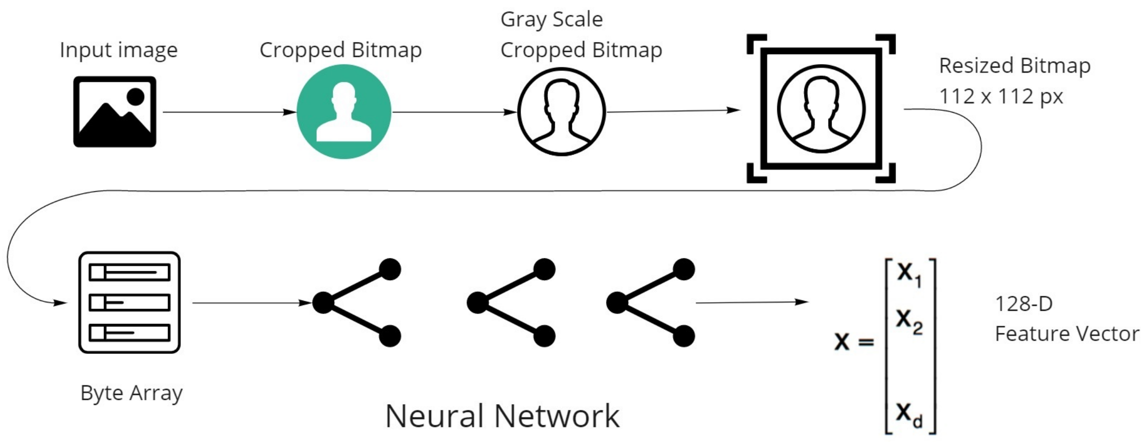

Input image
Cropped Bitmap
Gray Scale Cropped Bitmap
Resized Bitmap 112 x 112 px
Byte Array
Neural Network
$X = \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_d \end{bmatrix}$
128-D Feature Vector

Figure 3.2: Features Extraction

In the case of the enrollment, since we don't need to analyze the whole stream of frames, but just one of them, it has been decided to allow the user to take a picture that defines the frame with which the procedure can be executed.

The first step of the enrollment phase is face detection. So the single frame is given as input to the Face Detection model, which returns a list of detected faces. If the list contains more than one detection, the biggest one is chosen. For every detection, we have the coordinates of the face region into the input frame. After the detection phase, the input frame is converted into a Bitmap object, a standard way to represent images in Android. At this point of the procedure, the preprocessing phase begins. So the Bitmap is cropped to isolate the face from the rest of the image. Then the cropped bitmap is transformed into a grey-scale Bitmap and it is resized to 112 x 112 px, in order to be processed by the Face Recognition Model. In the end, the Bitmap is converted into a ByteArray and it is then ready to be processed by the Face Net Convolutional Neural Network. As said before, the output of the Face Net neural network is a 128-dimension vector that is finally memorized into the local memory of the Android device, with the related username. The memorized vector will be used in the recognition phase.

3.4 Recognition and antispoofing

In the Recognition phase, the way of managing frames is quite different. Indeed, in this case, it is necessary to process more than one frame, in order to perform the recognition. Therefore, when the user decides to start the procedure, it will be analyzed every frame captured by the OpenCV camera. The antispoofing system is a very important theme that affects the way the recognition procedure is designed. In the first version of the system, another CNN dedicated to the anti-spoofing task was used. This CNN uses the convolutional layers to underline specific aspects of the texture of the frame and representing it as a d-dimensions vector, with whom a binary classification could be performed (Spoofing/No spoofing).

Unfortunately, this would have increased the latency and the computational complexity too much, and because of the mobile environment, the frame rate of the whole procedure would have dropped down. That is why, in a second version of the system, another antispoofing approach has been used. The main idea is to make the user execute an easy random tasks sequence (close your right eye, smile, etc.) in a low time interval (10 seconds) in order to detect the liveness of that specific user. In this way, when the Face Detection phase is executed, the Face Detector model is used in such a way to retrieve additional information about the face detected. This information is used in the final part of the procedure, to decide if the user is executing the current task of the sequence or not.

After the face detection phase, if a face is detected, the frame is processed in the same way as the enrollment procedure. The frame is cropped, scaled to 112x112px and converted into a grey-scale ByteArray. Then, the image can be processed by the CNN that is used to produce the 128-dimensions vector. Once the vector is produced, it is compared with the embedding belonging to the claimed identity, enrolled previously. The comparisons between the vectors is executed by computing the euclidean distance: if the distance is below a fixed threshold, the recognition returns a positive result. The last part of the verification phase is the antispoofing control. Indeed it is verified if the recognized identity is actually performing the current task of the sequence (in order to detect the liveness) by using the information given by the Face Detector model. If the current task is detected on the verified identity, the sequence proposes the next task to perform. The procedure of verification ends when the user executes all the tasks of the random sequence before the expiry of the interval.

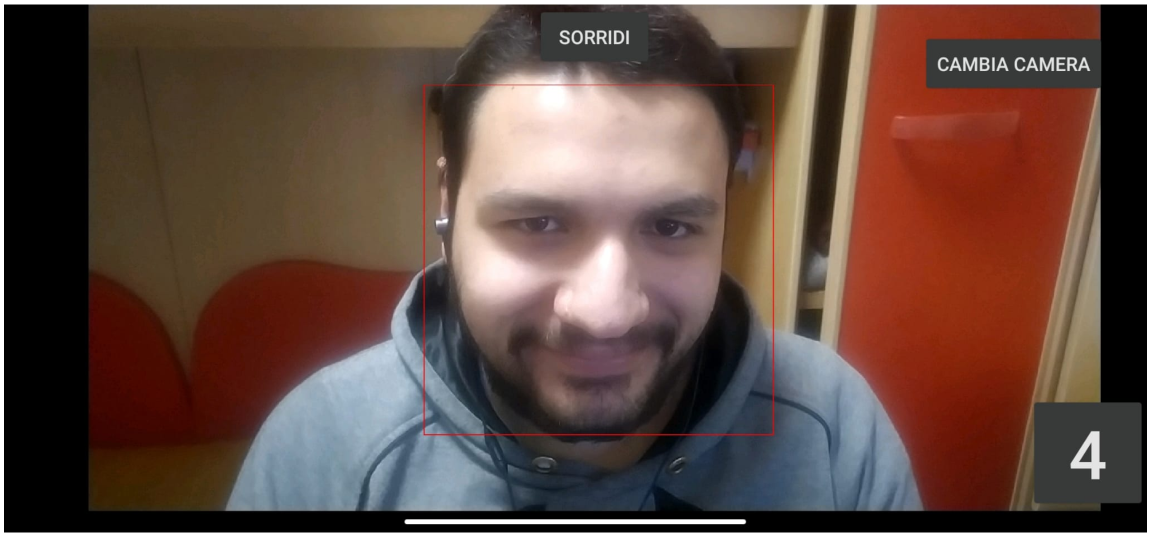


Figure 3.3: Recognition and antispoofing process with the ”smile” task

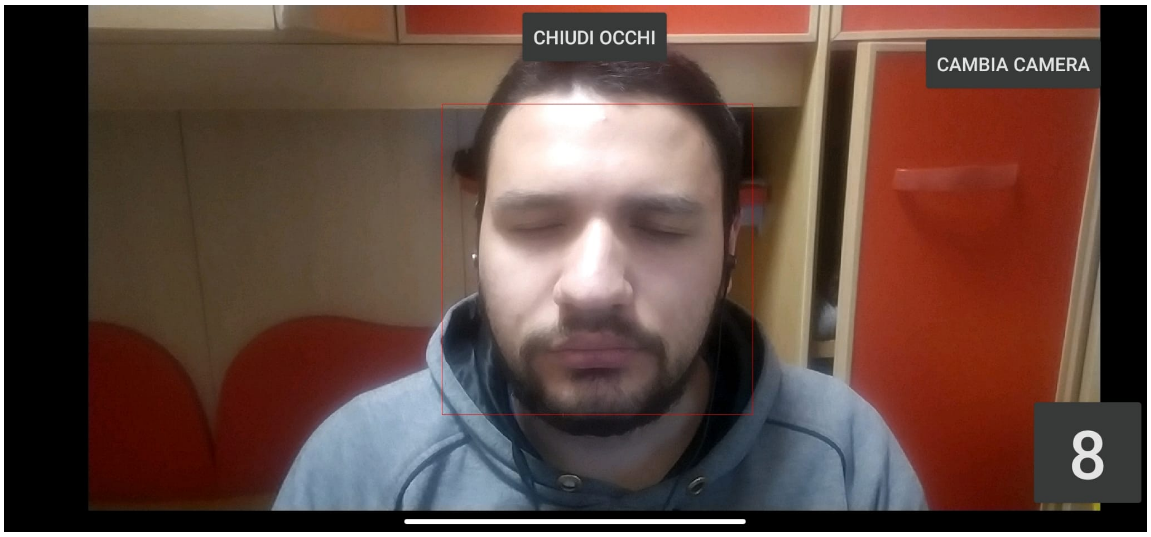


Figure 3.4: Recognition and antispoofing process with the ”close eyes” task

Chapter 4

Voice Verification

4.1 General Approach

The operations for the Voice Verification are executed both locally and remotely using the Azure APIs. The audio data are acquired locally with the phone microphone via the application and the data are passed to the Azure API.

In the Enrollment phase, the user records audio whose sum of durations is 30 seconds. After the enrollment, the user has a voice profile that will be used to verify the other audios that will be recorded when a transaction is performed.

To do this, the app records audio using m4a type with 48KHz of Sample Rate. Before the audio is passed to the API, it is converted since the Azure API only supports WAV audio with 16KHz of Sample Rate, 16 bit per mono PCM file.

If the verification phase returns a positive result, the user has been verified, otherwise no.

4.2 Work done on technology

The Voice Verification had two versions, in the first version we used a library called Recognito that converts audio files to double arrays.

The "fingerprint" of the profile is created when the first audio is passed to the system: it is converted in a double array, its features are extracted and stored. The following audios are used to deepen the profile voice.

After training, Recognito uses the Euclidean distance to recognize a person and returns a compatibility list with all registered persons. This version was shelved after the team has worked for a long time on file preparation to make this method safe, without great results.

The two problems that led us not to use this method are:

- We don't need to compare the voice with all people that are registered, but we need the verification of the single speaker of the wallet.
- The duration of the file deceived the system. In fact, if we used ten short audio files (5/7 sec each) to learn a voice of a person, after using a long audio (20 sec) to verify the voice this could be accepted even if it is not of the exact person.

That is why we moved to the second method that involves the use of the Azure API.

Azure provides many kinds of APIs; we have chosen to use the Text Independent Voice Verification. This means that the user can say any phrase and the system should be able to verify it. Azure works in this way: to use it, you must call your API by passing the files you want to use and all the attributes necessary for the operation, such as the token provided during registration and the user id. After this, the API returns a response. In the Registration API, the output contains the id of the new profile created and the creation date. In the Verify API, the response is composed of the similarity score of the audio recording (from 0 to 1), and "Rejected" or "Accepted" whether the user is verified or not (based on a default threshold of 0.5).

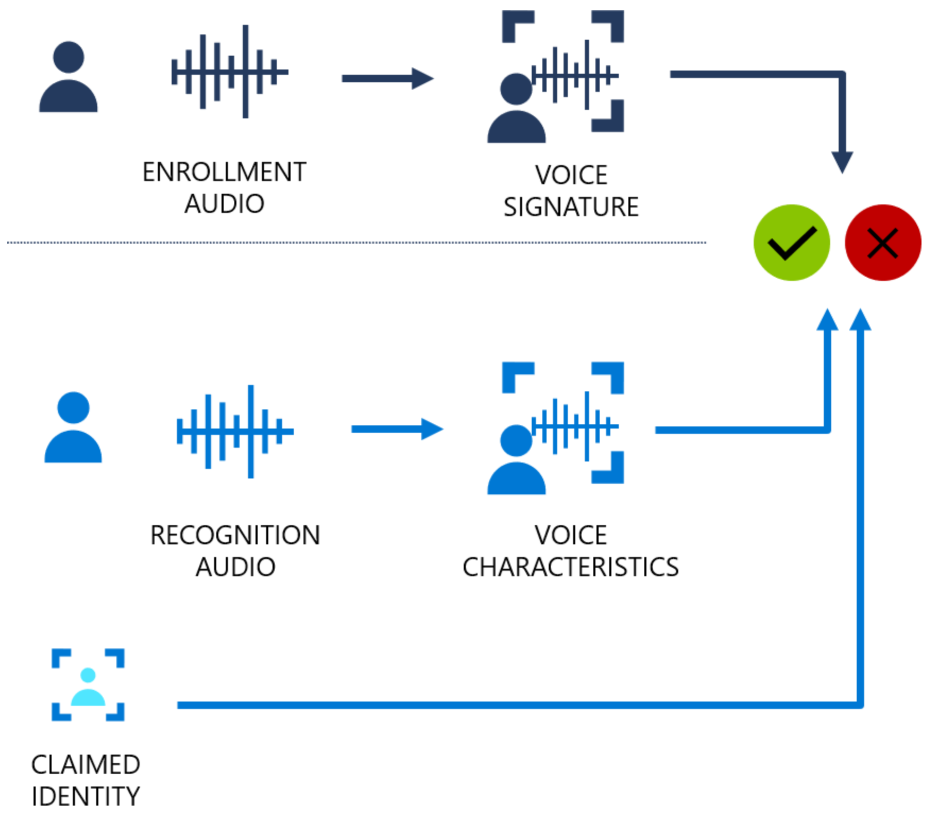


Figure 4.1: Enrollment/Verification flow

4.3 Storage and use of data

All the audios are not stored in our system. In fact, when a person records audio, it is passed to the API that extracts the features and uses them to register or verify the person. The only data that is saved is the Azure profile id that is provided from the API after the enrollment.

This means that we do not have any data of the user's voice but we can retrieve it. In the various Azure functions, the audios are processed and used in different ways. In the first API call, the Create Profile API, we pass the local id, consisting of language and country (i.e "it-it"). This API returns a profileId that refers to a new profile.

The second API is called the Create Enrollment API. It requires the profileId of the enrolled person inside the URL and the request the audio in binary form in the body of the request. The function returns the date of the creation of the Profile, the number of audio enrolled, the enrollment status and the time remaining before the profile is enrolled.

The third API that we use is called Verify Profile. Like the previous one, it takes the profileId of the person that we need to verify the identity of. The body of the request will contain the binary file audio of the user who wants to make the transaction. The API processes the audio and compares it with those it has already acquired. If the user is verified, The response is "Accepted" otherwise "Rejected".

Chapter 5
Evaluation

Since the user who is trying to access claims a username and a password, the system performs a verification task. In particular, the subject is accepted if the distance (the similarity) achieved from matching with the gallery templates corresponding to the claimed identity is $\leq$ ($>$) the acceptance threshold, otherwise it is rejected. Therefore, we have to find a threshold that maximizes the performance, computed through the number of Genuine Acceptance (GA), False Rejection (FR), False Acceptance (FA) and Genuine Rejection (GR).

Of course, since the system uses two different methods of authentication, it is necessary to split the evaluation task into two parts.

5.1 Facial Recognition

In order to test the performance of the facial recognition, we created a little dataset composed of some photos of us to which we added some impostors pictures.

Our first goal was to use some big dataset of several people collected by universities or other research teams downloadable from the internet. There are plenty of studies that focus on faces (not only for recognition) and a little part of them shares the collected photos. However, we were interested only in those datasets that fitted our use-case. The future adopters of the app are going to take photos of them for enrollment or verification from the front camera which generally have a very low quality (or a worse quality than the retro one, at least), with random poses, in situations that have not necessarily good light conditions. Of course, the faces could also take different poses, but we can expect that the most common is going to be the frontal one because the user needs to interact with the app during the process.

After some researches, the only two datasets we were able to find that satisfied some of our requirements are the following:

- KomNET [1]
- UMDAA-02 [2]

They both contain tens of users with several types of photos in quite different situations. The UMDAA-02 is also composed only of photos taken by themselves from the front camera. However, the photos in the former were all already cut, leaving very little work for our face detection system. Moreover, photos were quite unnatural and the expressions differed too little. The tests with this dataset have shown very good results, the system was correctly verifying all the users also with very low values of thresholds, even if our empirical attempts ruled out this possibility. The latter instead was very close to what we were searching: hundreds of photos for each user, all taken by the front camera in different situations, with different light conditions, poses and expressions. However, the photos in this dataset are not classified, therefore a folder can contain also photos with no faces or someone else's. Moreover, the majority of photos were very blurry and/or cut (face not completely visible) and they obviously could not be useful to test the real performance of the system. Finally, the necessary time to correctly classify the photos was too high to make it worth it.

5.1.1 Creation of face dataset

Therefore, we ended up creating our own dataset, which obviously could not be large like the previous ones but which had the main advantage to meet all our requirements. The lower number of enrolled users, indeed, is not going to be a problem because our app works locally on the phone of the user, therefore we cannot expect more than a couple of users enrolled on the same device. We are going to enrol four people at the same time.

In particular, each member of the group took sixteen photos from the front camera of his own smartphone with different poses, lights and expressions. One of us also kept his glasses on that reflect the phone screen. Using one of these photos for enrollment, we did fifteen genuine attempts for each user.

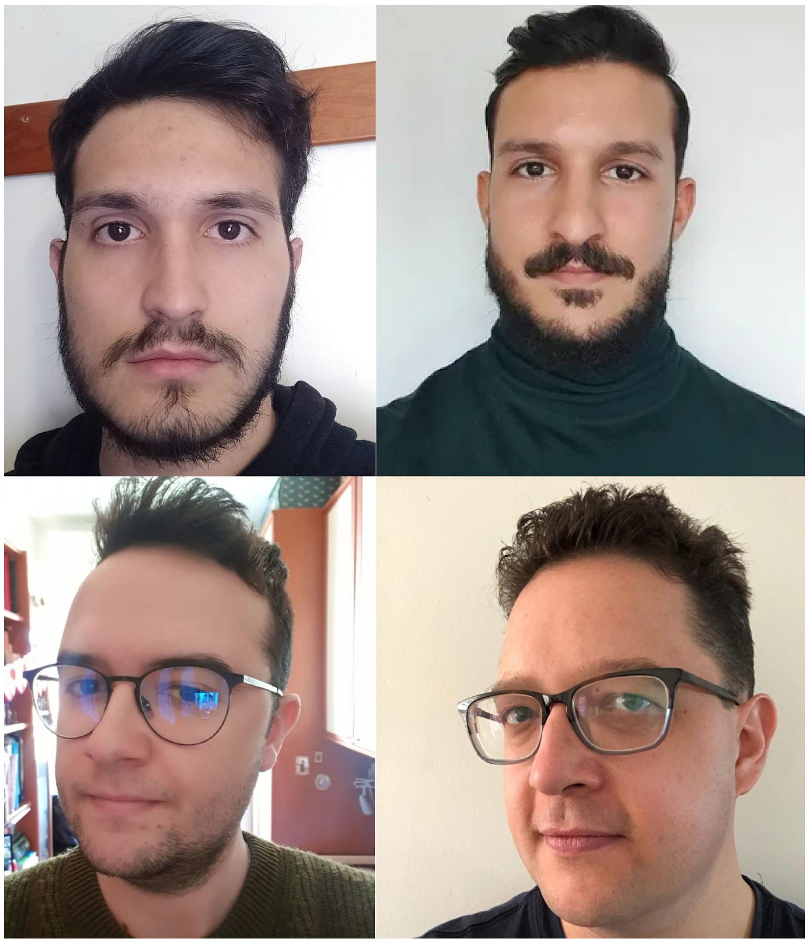
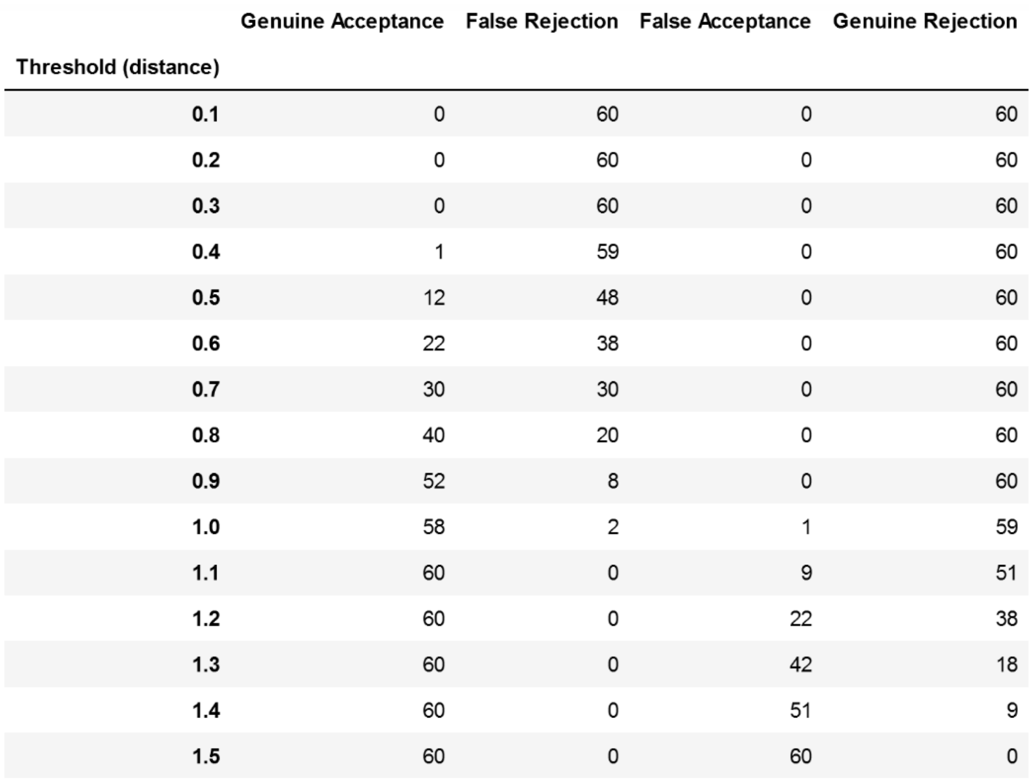


Figure 5.1: On the left, two photos of genuine users. On the right, two of the impostors selected through the reverse image function

Of course, we also need some non-genuine attempts in order to get the number of FA and GR. We created a folder of impostors containing one random photo of each member of the group plus three photos of his "lookalikes", found online using reverse image function on some search engines. Then, the total number of impostor attempts is going to be: 4 photos of ours + 3 lookalike photos * 4 members - 1 photo of the user being evaluated = 15 impostor attempts for each user = 60 impostor attempts.

In this way, we are going to have an equal number of genuine and non-verification requests for each round.



	Genuine Acceptance	False Rejection	False Acceptance	Genuine Rejection
Threshold (distance)				
0.1	0	60	0	60
0.2	0	60	0	60
0.3	0	60	0	60
0.4	1	59	0	60
0.5	12	48	0	60
0.6	22	38	0	60
0.7	30	30	0	60
0.8	40	20	0	60
0.9	52	8	0	60
1.0	58	2	1	59
1.1	60	0	9	51
1.2	60	0	22	38
1.3	60	0	42	18
1.4	60	0	51	9
1.5	60	0	60	0

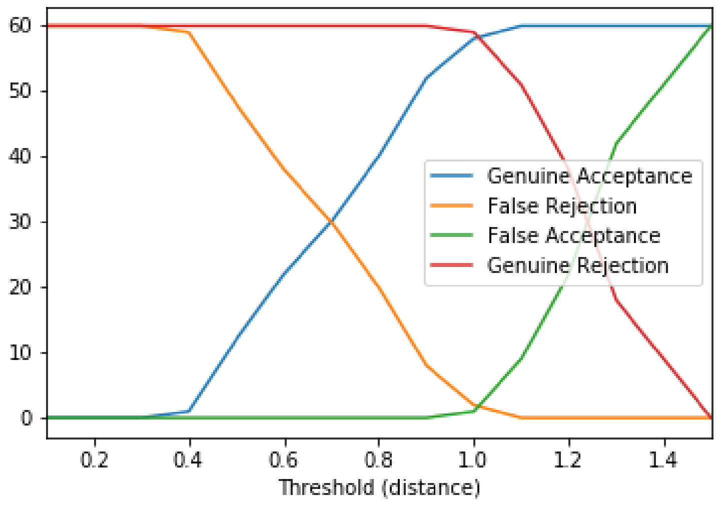
5.1.2 Results

We wanted to study the behaviour of the system as we increase the threshold. The possible values were those in the range from 0.1 to 1.5 with 0.1 steps. The maximum value was decided after empirical attempts which showed that larger values would correspond to trivial results (every image containing a face was correctly verified, whatever was the claimed identity).

After the enrollment of the four users, for each value of the threshold t, 120 attempts were done considering the threshold t. Of course, we expect the FR and GR to be at the maximum at the beginning and tend to zero as the threshold increases. We want to find the value of the threshold which maximizes the number of GA and GR.

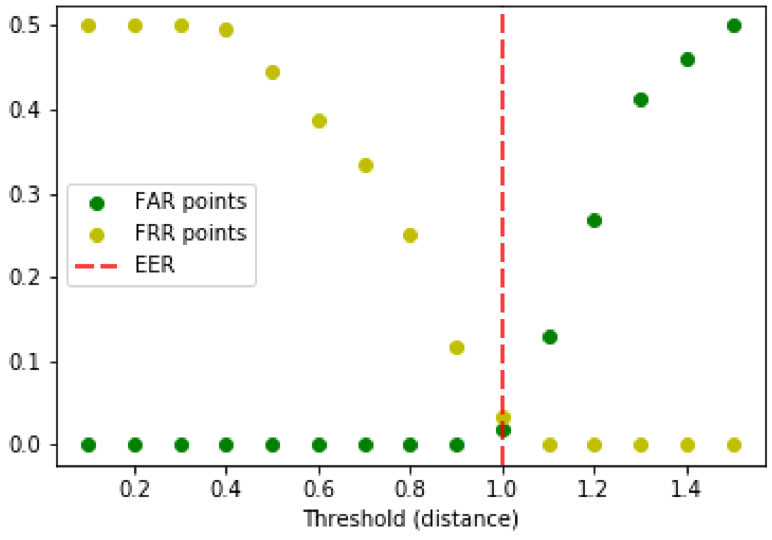
The results obtained with the dataset described above are summarized in the table of Figure 5.2.

As we can see, the behaviour is like expected and the extreme values of the threshold (0.1 and 1.5) correspond to the worst results of the metrics. However, we can start to notice that, for some value of the threshold, the number of wrong recognition (FA + FR) is close to 0. Let's plot the results in a cartesian plane in order to get a better overview of the trends.



False Rejections (yellow line) starts to decrease with values of threshold greater than 0.3 and it reaches its minimum value when the threshold is equal to 1.1. We can observe an exactly opposite trend for the Genuine Acceptances (blue line); the number of GAs becomes 60 for $t = 1.1$. However, we cannot say that it is the optimal value since the numbers of Genuine Rejections and False Acceptances start to decrease (and increase, respectively) for $t = 1.0$. Moreover, using a threshold of 1.1 would mean having no false rejection but nine false acceptance. Since the consequent verification is allowing access to a digital wallet, we cannot allow such a large number of false acceptance.

In the following figure, you can observe the False Acceptance Rate (FAR) and the False Rejection Rate (FRR) plotted against the different values of the threshold, and the vertical axis corresponding to the Equal Error Rate (EER), which is placed at $t = 1.0$ since in that point the FAR and the FRR intersect.



Nevertheless, in our use-case, it is more important to prevent access to the system by impostors instead of some wrong rejection which would result just in one more attempt by the genuine user. That is why, even if the ERR lays on the 1.0 threshold and the number of total errors would be three instead of eight, we eventually selected 0.9 for the threshold. It means that the system is going to wrongly classify eight probes out of sixty but still the number of FA is going to be equal to zero.

5.2 Voice Verification

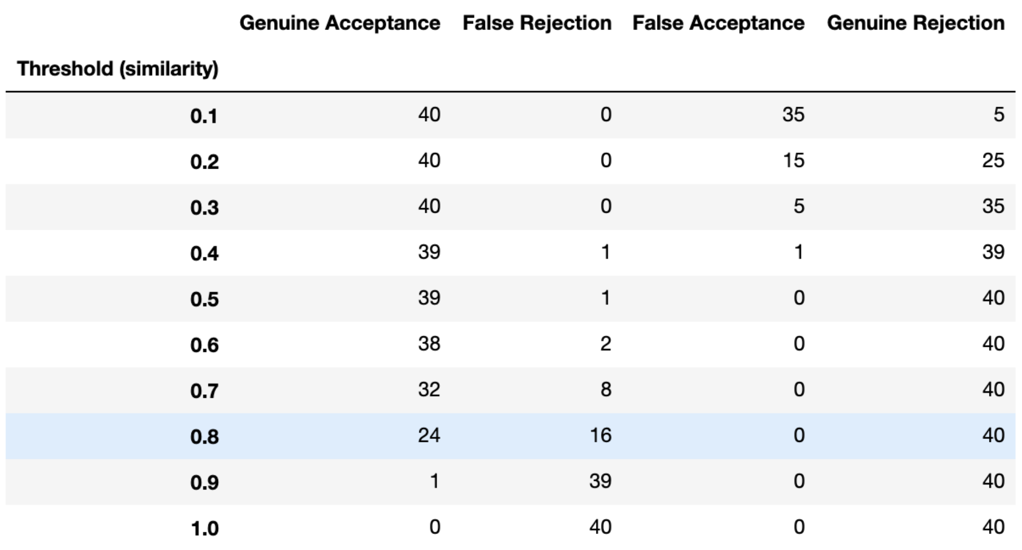
Also the dataset used to evaluate the voice verification was created from some audio of us to which we added some impostors audio.

Since we do not have a main phrase that a user can say to interact with our system, all audio files are different in length and content. Since the app works locally, we do not need a large number of enrolled people to test the system. We assume that on the same device there will not be too many accounts saved. We are going to enrol four people at the same time.

In detail, each member of us recorded 12 audio with an average length of 10 seconds. Two of these audios are used to enrol the person. The other 10 audio are used to have genuine attempts for each user. Of course, we also need some non-genuine attempts in order to get the number of FA and GR. We created a folder containing audio of impostors. The total number of impostors attempt is going to be 10 impostors audio * 4 members = 40 impostor attempts. In this way, we are going to have an equal number of genuine and non-verification requests for each round.

5.2.1 Results

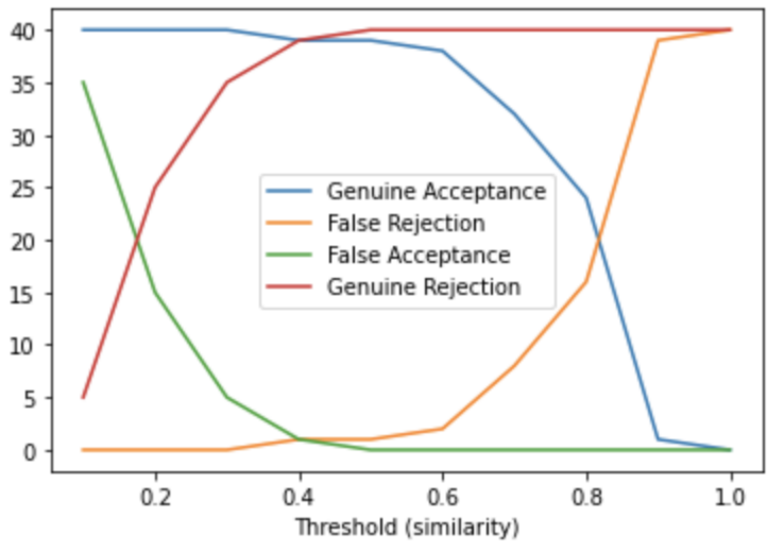
In this case, the possible values of the threshold were in the range from 0.1 to 1.0 (still with 0.1 steps) and correspond to the similarity instead of the distance. Since the threshold of the API cannot be set manually, the results of the 80 attempts were produced taking into account the scores and simulating the accepts or rejects with the different thresholds.



	Genuine Acceptance	False Rejection	False Acceptance	Genuine Rejection
Threshold (similarity)				
0.1	40	0	35	5
0.2	40	0	15	25
0.3	40	0	5	35
0.4	39	1	1	39
0.5	39	1	0	40
0.6	38	2	0	40
0.7	32	8	0	40
0.8	24	16	0	40
0.9	1	39	0	40
1.0	0	40	0	40

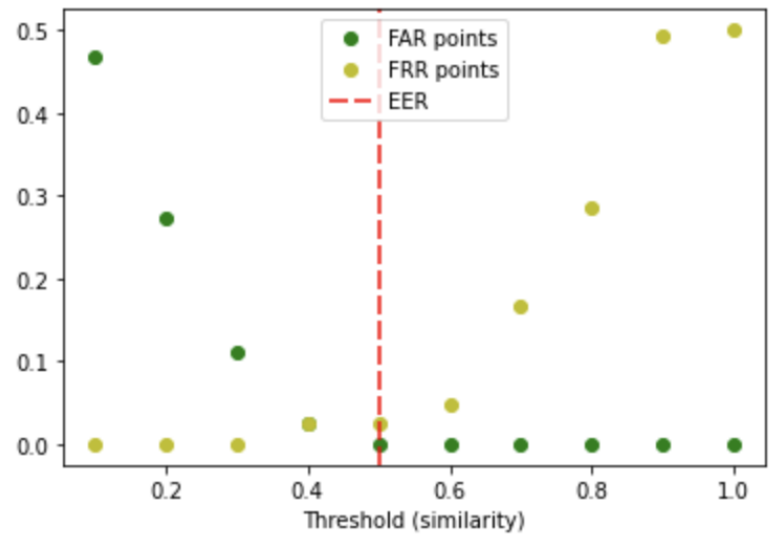
Figure 5.2: Detailed table for GA, FR, FA, GR

The value of the threshold is the minimum value of similarity that we use to accept a user. As we can see in the Figure 5.2, the number of wrong recognition is close to 0 for some values of the threshold.



In the above plot, we can observe that False Rejections (yellow line) starts to increase with values of threshold greater than 0.4 and it reaches its maximum value when the threshold is equal to 0.9. The trend is exactly the opposite for the Genuine Acceptances (blue line); the number of GAs is 40 for $t = 0.3$. However, we cannot say that it is the optimal value since the numbers of Genuine Rejections and False Acceptances start to increase (and decrease, respectively) for $t = 0.3$. Moreover, using a threshold of 0.4 would mean having no false rejection but one false acceptance. Since the consequent verification is allowing to send money, we cannot allow false acceptances.

In the following figure, you can observe the False Acceptance Rate (FAR) and the False Rejection Rate (FRR) plotted against the different values of the threshold, and the vertical axis corresponding to the Equal Error Rate (EER), which is placed at $t = 0.5$ since in that point the FAR and the FRR intersect.



As we said before, it is more important to prevent access to the system by impostors instead of some wrong rejections, because it is sufficient that the genuine user records another audio to perform a new attempt. The threshold that has been eventually selected is 0.5, which corresponds to the default threshold of the Azure API. It means that the system is going to wrongly classify about one probe out of forty but still the number of FA is probably going to be equal to zero.

Bibliography

[1] Nyoman Gede Arya Astawaa et al. *KomNET: Face Image Dataset from Various Media for Face Recognition*. 2020. URL: https://www.sciencedirect.com/science/article/pii/S2352340920305710.

[2] Rama Chellappa et al. *Active Authentication on Mobile Devices*. 2015. URL: http://users.umiacs.umd.edu/~rama/research.html.