M.Sc. in Computational Data Science

Samuele Ceol (13140)

Capstone Project Report

B-GAN: A multimodal smartphone-based authentication system

February 2023

Faculty of Computer Science

Abstract

The following report describes the main components of B-GAN, A multimodal smartphone-based authentication system developed as part of the Capstone Project activity carried out in the context of the Msc. in Computational Data Science at the Free University of Bozen-Bolzano in the period between August 2022 and January 2023. The content of this document (together with the additional resources linked to it) seek to describe the set of activities involved in the development (and subsequent testing) of the presented application. In this context, the report is prefaced by a general introduction section describing the goal of the application and contextualising the previous work on which B-GAN was built on (Chapter 1). Then, a description of the technical information and underlying code architecture is provided. In this regard, the relevant code repository and developer documentation are also provided as external links (Chapter 2). In Chapter 3, the high level workflow and main UI elements related to the application are introduced and described. Chapter 4 describes the structure of the individual machine learning models (and related features) that ultimately make up the presented authentication system. Finally, Chapter 5 provides the details related to the training and testing procedure and describes how the relevant data was gathered from the end users. Chapter 6 presents the results stemming from the data gathering phase. Finally, the last chapter of this document (Chapter 7) is dedicated to mentioning some possible avenues for future work activities.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Behavioural biometrics is the research field concerned with measuring and analysing human behaviour with the purpose of identifying specific patterns that can be used to build unique user profiles. These patterns, which derive from the way in which a user interacts with a given system, can be used as means for authentication. Unlike physiological biometrics, where authentication systems try to capture physical characteristics of the user via dedicated sensors (e.g. fingerprint, iris, etc.), behavioural biometrics authentication systems do not require any additional dedicated hardware in order to gather the data needed to build a given profile.

This work introduces B-GAN, a multimodal smartphone-based user authentication system presented in the form of an Android application. B-GAN uses up to three different input mechanisms (a swipe gesture, a PIN code and a signature) together with additional information related to the user's holding behaviour in order to gather the behavioural data required to build a set of machine learning models which are used for authentication purposes. While using B-GAN, a user can provide a set of training interactions which are used by the system to obtain the data necessary to observe its usage patterns. Given the fact that the application uses this data to train a (set of) one-class classifiers, only positive samples from a genuine user are required in order to build these models. This means that no fraudulent interactions need to be gathered during the training procedure. Once the active models are successfully built, the same user (or another one impersonating an attacker) can test the accuracy of the system by providing one or more test interactions and observing the responses of the application. Detailed results relating to a given user and stemming from training and testing the models are stored on the application's DB and can be saved and easily exported in the form of a set of .csv files. Such results not only include the performance relating to the generated models, but also granular information on the individual interactions used for building them. Additionally, the application allows researchers to exonerate specific features and even entire models from the training procedure with the purpose of easily observing how disabling certain sources of information can affect the overall accuracy of the system. In the context of the presented application, a generative adversarial network can also be trained and utilised to generate synthetic samples which can contribute to the training procedure. B-GAN is a stand-alone application that performs all training steps locally using solely the hardware resources provided by the mobile device at hand.

1.1 SwipeGAN

This work represents an extension built on the initial functionalities presented in SwipeGAN, a development work originally carried out by the students Marco Di Panfilo and Gianluigi Pelle in the months of June-September 2021 in the context of the Capstone Project offered by prof. Attaullah Buriro at the Free University of Bozen-Bolzano.

SwipeGAN proposed a unimodal authentication mechanism that made use of data collected from a set of swipe gestures together with information gathered from the mobile device's accelerometer and gyroscope sensors in order to build a single model (containing all the observed features) used to classify (accept or reject) further swipe interactions.

B-GAN builds on top of these functionalities (and related software architecture) to introduce a new set of features and interaction modalities turning the application into a multimodal authentication system, where the different features are organised into various categories making up the individual models. In this context, B-GAN allows for multiple models (and model combinations) to be built during the training procedure by concatenating features contained in the individual models or by making use a weighted ensemble where the individual models take part in a voting procedure to accept or reject a given interaction. Additionally, this work introduces a new model profile view that allows to easily tweak various settings related to the individual models, including choosing the active features and the model combinations that are built during the training procedure.

In addition to these application-level differences, other major activities that were carried out in the context of this work include the production of the complete documentation related to the final codebase associated with the application and a data gathering phase where interactions were collected from real users in order to gain insights on the performance of the system.

In addition to the aforementioned extensions, a list of other minor changes and relevant fixes that have been carried out on the original codebase are mentioned in the Appendix 7.1 section that can be found at the end of this report.

Chapter 2

Technical Information

Metadata description	**Information**
Link to code/repository	https://github.com/iberius96/B-GAN
Code versioning system used	git
Software code languages, tools, and services used	Java, Android Studio
Link to developer documentation	https://iberius96.github.io/B-GAN
Support email for questions	samceol@unibz.it

Table 2.1: Techinical information

The technical description and the related resources for the presented work are summarised in Table 2.1. Accessing the developer documentation is encouraged in order to get a general idea of the architectural structure of the presented software. Additionally, note that the linked documentation contains descriptions of all the code components that are part of the application. This includes:

- Classes.
- Class variables.
- Methods and the related parameters.

2.1 Software architecture overview

The main overarching architectural components of the B-GAN application are briefly described as follows:

1. The `MainActivity` class handles the vast majority of the core logic of the application. It allows for the correct operation of the UI elements that are part of the data collection, training and testing process, it ensures the correct collection of data originating from the device touch screen and sensors, it handles the interactions, classifiers and GAN objects and it uses said objects to carry out the training and testing activities.

2. The `DatabaseHelper` class allows the application to persist all information related to the data collection process for a given user. This is done by means of an SQLite database which stores all the data that can ultimately be exported in the form of a set of .csv files (values for the individual interactions, information on the active features, user details, etc.).

3. The `Swipe` class is used to model the individual interaction object that, upon completion, is stored in the DB.

4. The `GAN` class represents the adversarial network and contains the logic required to setup and train the network and to ultimately trigger the generation of the synthetic samples.

In addition to the aforementioned components, the following elements are designed to handle activities that are part of some of the application's secondary routines:

1. The `ProfileActivity` and `ModelActivity` classes handle the contextual population of UI elements related to the user and model profile views. Upon closure of the respective views, these classes also update the modified fields on the active DB.

2. The `ResourceMonitor` class handles the collection (during the model's training procedure) of the hardware resources utilised by the mobile device (core CPU frequency, memory usage and total power draw).

3. The `SignatureView` class handles the logic tied to the signature's canvas.

4. The `RawDataCollector` class allows for continuous collection of interaction's data following a specified frequency. Unlike the traditional data collection process (where one DB entry is generated for each interaction), the `RawDataCollector` allows to treat the sensor's data as a continuous stream.

Note that the data collected by the `RawDataCollector` is stored on the DB but not currently used in the training procedure for any of the generated models. This functionality of B-GAN can be seen as an initial setup for future work that might be carried out in the context of this project.

Chapter 3

The Application

B-GAN stores genuine interactions' data gathered from a given user and uses it to generate a set of models capable of accepting or rejecting further interactions received by the system based on their individual characteristics. The idea is that the specific ways in which different users interact with the mobile device should allow B-GAN (and the generated models) to successfully discern a genuine entity by a potential attacker. The overall process that can be performed on the application to test this principle is highlighted in Figure 3.1, the adversarial network and the synthetic interactions are represented using dotted lines since they are non-mandatory entities when utilising the application. The application will require a minimum of five train interactions to build the models and at least one test interaction to test them. Naturally, gathering more data will result in more accurate models and performance metrics. The performance metrics are represented as a separate entity from the application since they are meant to be exported (as a set of .csv files) and analysed outside of the mobile device. Generating the full set of performance metrics for the classifiers at hand will require for both genuine and fraudulent test interactions. However, B-GAN will allow the user to complete the testing procedure even if only one of the two types of interactions has been gathered.

The following sections will provide a description of the user interface elements (i.e. the main views) that are part of the application. This set of descriptions will roughly follow the class structure highlighted in Section 2.1 and should give a general idea of how the high level architecture of B-GAN has been layed out.

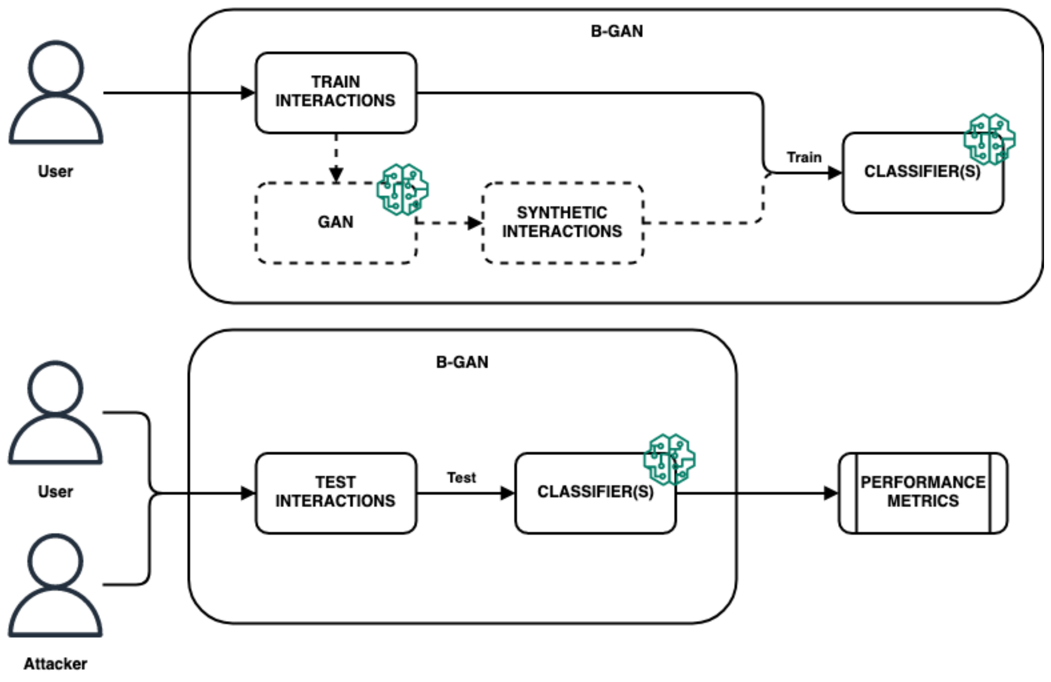


Figure 3.1: B-GAN training & testing workflow

3.1 Main view

The main view encapsulates the vast majority of the user interface elements that are part of the core services offered by B-GAN. Firstly, the main view allows to collect interactions as part of the training procedure. Secondly, it allows for the models' training to be triggered (with or without the generation of the adversarial network) using the *TRAIN* and *GAN* buttons. Finally, from within this view it is also possible to erase the data contained in the current DB (using the *RESET DB* button) and to access the user and model profile pages (via the *PROFILE* button). Additionally, once the models are built and the training procedure is concluded, the main view presents the interface elements required to test the accuracy of the system and to ultimately save the set of collected results.

3.1.1 Train interactions

In the context of B-GAN, a train interactions is made up of (up to) three different gestures that are performed by the end user in direct sequence. The first gesture is represented by a single (uninterrupted) swipe across the touch screen. At this stage, a stylized image of the gesture is presented as reference to the user. Once the swipe gesture has been completed, the user is presented with an 8-digit numpad where the PIN-code can be inserted. Upon completion of the keystroke gesture, the application automatically transitions to the signature view. In this context, the user is presented with a blank canvas where the signature gesture can be drawn. Additionally, the user is able to delete the currently drawn signature using the *CLEAR* button and to proceed to the following interaction using the *NEXT* button. The overall process described in this is visualised in Figure 3.2. Train interaction can (and generally should) be recorder by the user using different different body postures. In this context, the application allows to mark a given (set of) interactions as being recorded either while sitting, standing or walking. Although this information is not used by the models during training (since ideally the classifiers should be able to identify a user independently from the body posture used when providing an interaction), it can still provide a useful insight with regards to how a given data collection process was carried out.

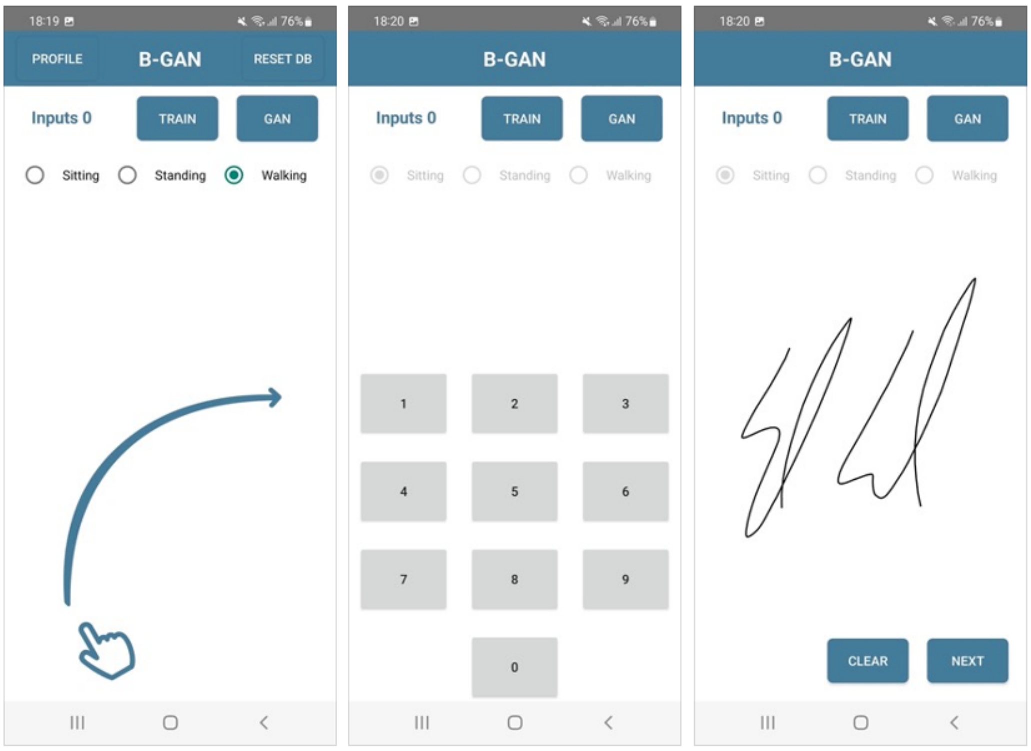
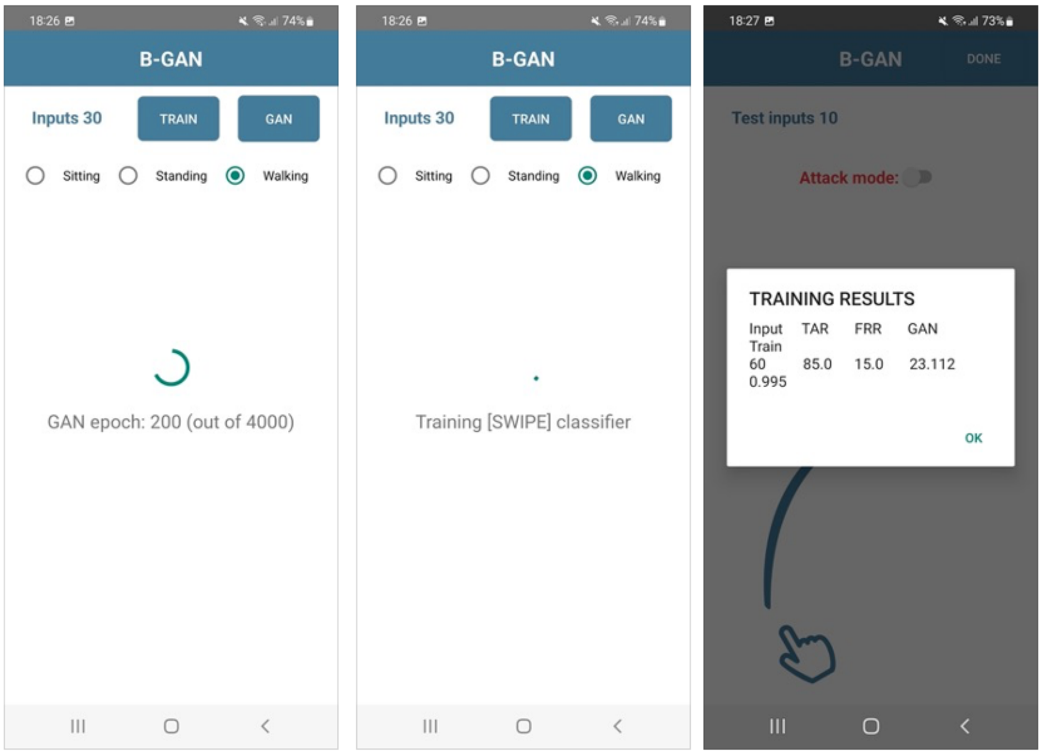


Figure 3.2: Views for a single interaction

3.1.2 Training

Once at least five complete interactions have been collected from a given user, the application will allow for the training process to be started. To do so, the *TRAIN* and *GAN* button can be utilised to initialise the procedure. The difference between these two training modalities resides in the fact that the one triggered using the *GAN* button will also train the adversarial network and use the synthetic samples as part of the training process. The number of generated synthetic samples will always be equal to number of genuine ones collected from a given user. In the context of the GAN training procedure, both genuine and synthetic samples are used to train the classifiers. The application will then build a set of models in accordance with the models combination setting specified in the model profile. The training of the adversarial network and the subsequent creation of the set of one-class classifiers is visually highlighted in Figure 3.3.



Sitting Standing [✓ Walking]

GAN epoch: 200 (out of 4000)

Training [SWIPE] classifier

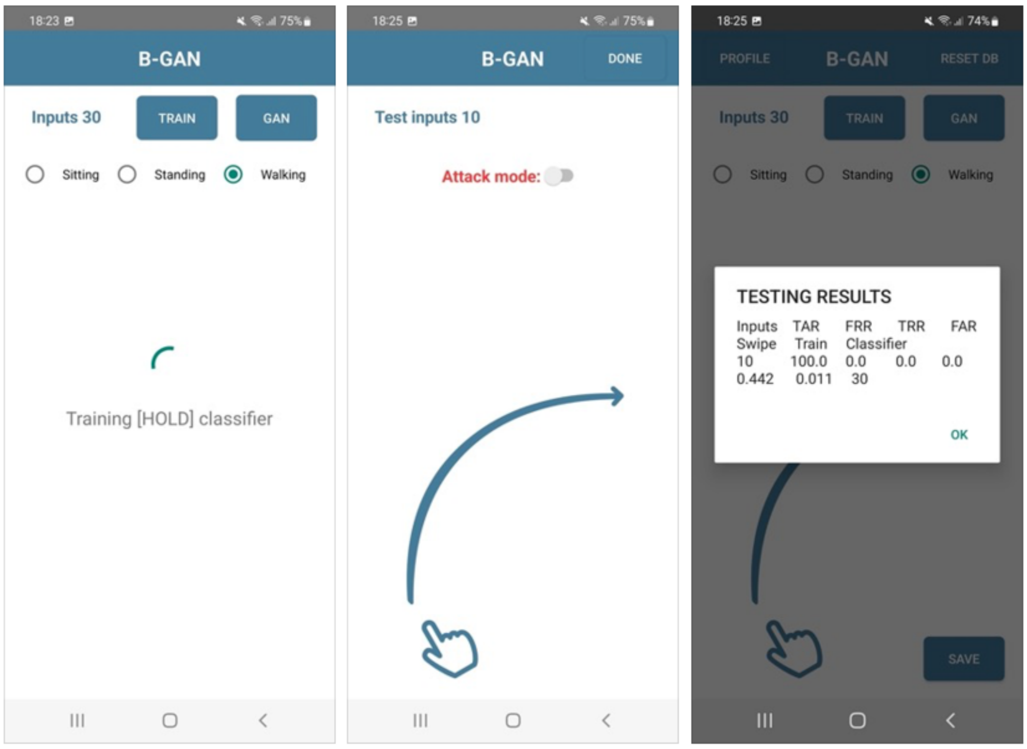
TRAINING RESULTS

Input	TAR	FRR	GAN	Train
60	85.0	15.0	23.112	0.995

Figure 3.3: Views related to GAN training

3.1.3 Test interactions

Upon completion of the training procedure, the user (or a potential attacker) will immediately be prompted to provide the test interactions required to asses the accuracy of the generated models. In this context, a test interaction will present the same overall characteristics of the training ones (i.e. if a specific feature or an entire gesture is disabled during training, it will also be disabled during testing). The user needs to provide at least one interaction in order to complete this particular step. Additionally, the *Attack mode* switch can be used to mark certain interactions as being provided by a non-genuine user that is attempting to infiltrate the system. For interactions provided in such a way, a rejection will be considered as the correct response from the models. Screenshots from the training procedure are presented in Figure 3.4.



```
TESTING RESULTS
Inputs   TAR     FRR      TRR     FAR
Swipe    Train   Classifier
10       100.0   0.0      0.0     0.0
0.442    0.011   30
```

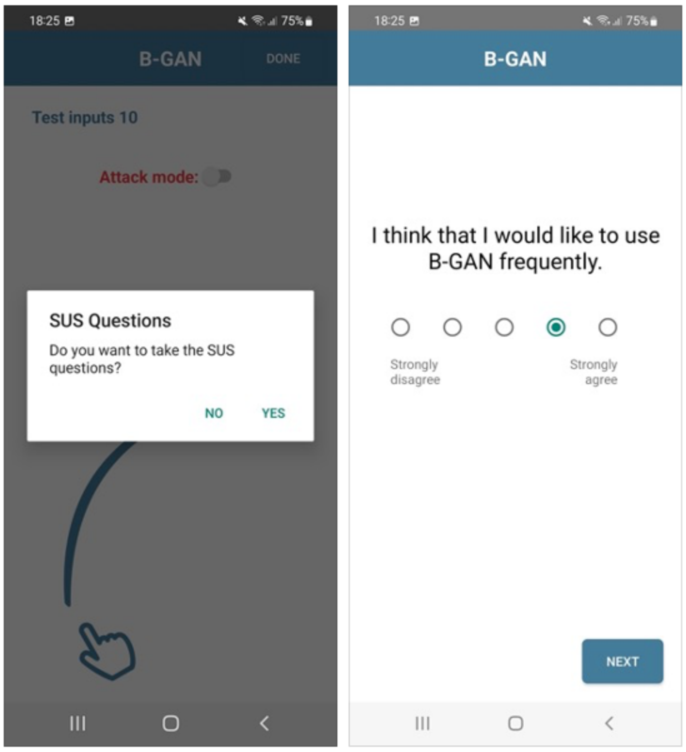
Figure 3.4: Views highlighting the training process and subsequent testing

3.1.4 System Usability Scale

In order to gather some information with regard to the perceived quality of the presented application, a set of statements based on the System Usability Scale can be presented to the end-user upon completion of the training phase. Each of this statements can be associated with a feedback based on a five-level likert scale representing the level of agreement of the user with regards to a given statement. The SUS phase is represented in Figure 3.5. The ten statements are presented as follows:

- I think that I would like to use B-GAN frequently.
- I found B-GAN unnecessarily complex.
- I thought B-GAN was easy to use.
- I think that I would need the support of a technical person to be able to use B-GAN.
- I found the various functions in B-GAN were well integrated.
- I thought there was too much inconsistency in B-GAN.
- I would imagine that most people would learn to use B-GAN very quickly.

- I found B-GAN very cumbersome to use.
- I felt very confident using B-GAN.
- I needed to learn a lot of things before I could get going with B-GAN.


SUS Questions

Do you want to take the SUS questions?

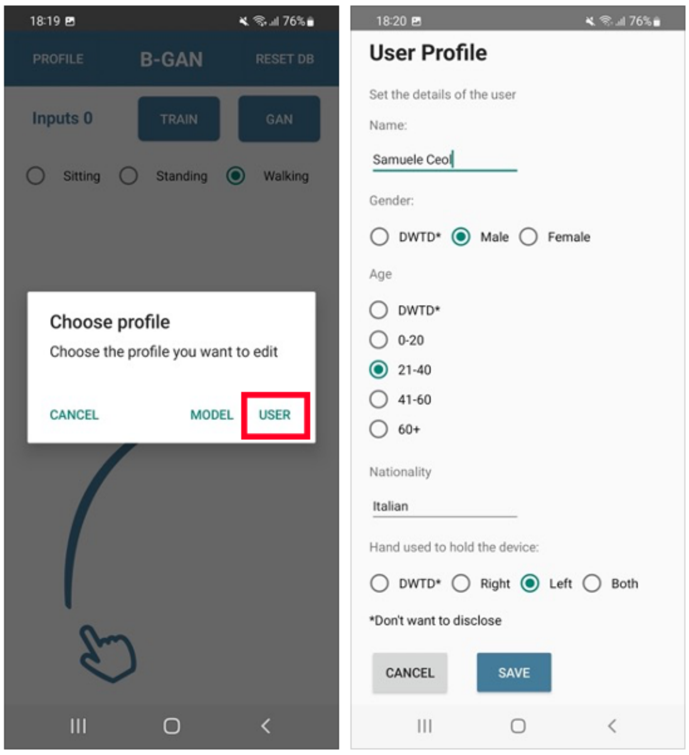
I think that I would like to use B-GAN frequently.

Figure 3.5: SUS questions views

3.2 User profile

The user profile view allows to define some general information related to the user that is currently testing the system. In this context, B-GAN provides a set of fields that can be field to indicate the user's (nick)name, its gender, the age range, the nationality and to specify which hand will be used to hold the device when collecting the samples. Naturally, all this information are fundamentally optional (hence the presence of the *Don't want to disclose* field) but can potentially be useful to provide further insights related to how factors such as age could influence the overall accuracy of the system.

The details of the user profile view and the dialog used to access it are represented in the screenshots in Figure 3.6.



Inputs 0

Sitting Standing [✓ Walking]

Choose profile

Choose the profile you want to edit

User Profile

Set the details of the user

Name: Samuele Ceol

Gender: DWTD* [✓ Male] Female

Age: DWTD* 0-20 [✓ 21-40] 41-60 60+

Nationality: Italian

Hand used to hold the device: DWTD* Right [✓ Left] Both

*Don't want to disclose

Figure 3.6: User profile views

3.3 Model profile

The model profile views provides a wide array of settings allowing to control the details of the constructed models. In this context, researchers can toggle the various individual features that are part of the interested models.

For the *SWIPE* model, B-GAN offers the possibility to decide whether to use data related to the duration of the gesture, the size of the screen contact area (i.e. the finger's size), the gesture's start and end positions and the velocity of the gesture. Additionally, researchers can also decide to collect the overall shape of the swipe and the number of segments utilised gather such information (details related to this segmentation technique are described in Section 4.1). With regards to the *HOLD* model, B-GAN offers the possibility control which of the three main components (acceleration, angular velocity and orientation) and corresponding sensors are considered for the collected samples. In the *KEYSTROKE* section of the view, B-GAN offers the possibility to enable the individual press durations and the keystroke intervals features. Additionally, in this section it is also possible to control the overall length of the PIN code (from 4 to 8 digits). Finally, the *SIGNATURE* allows to control the velocity, start and end position features of the signature gesture. Similarly to the swipe gesture, it is also possible to control the details related to the signature's shape features.

Additionally, the model view also offers the possibility to completely disable the KEYSTROKE and SIGNATURE gestures.

It is also important to notice that, in a general sense, disabling a given (set of) features won't prevent the application to gather the data related to it. In fact, in the context of B-GAN disabling a feature from the model profile view only prevents it to be used when training and testing the models. In other words, the application will still collect the full interaction data independently from the current model profile settings. The notable exception this fact is when a given gesture is completely disabled and therefore not accessible when gathering the training interactions.

From this view, researchers can also decide which (combination of) models should be constructed once the training procedure has been started. In this context, B-GAN presents the following options:

- *Full*, representing an individual model trained on the full set of features gathered for a given interaction across all sensors.
- *Individual + Full*, where the full model is built together with the four individual ones (Hold, Swipe, Keystroke and Signature).
- *All combinations*, where the full model, the individual ones and all their possible combinations are built when the training procedure is started.

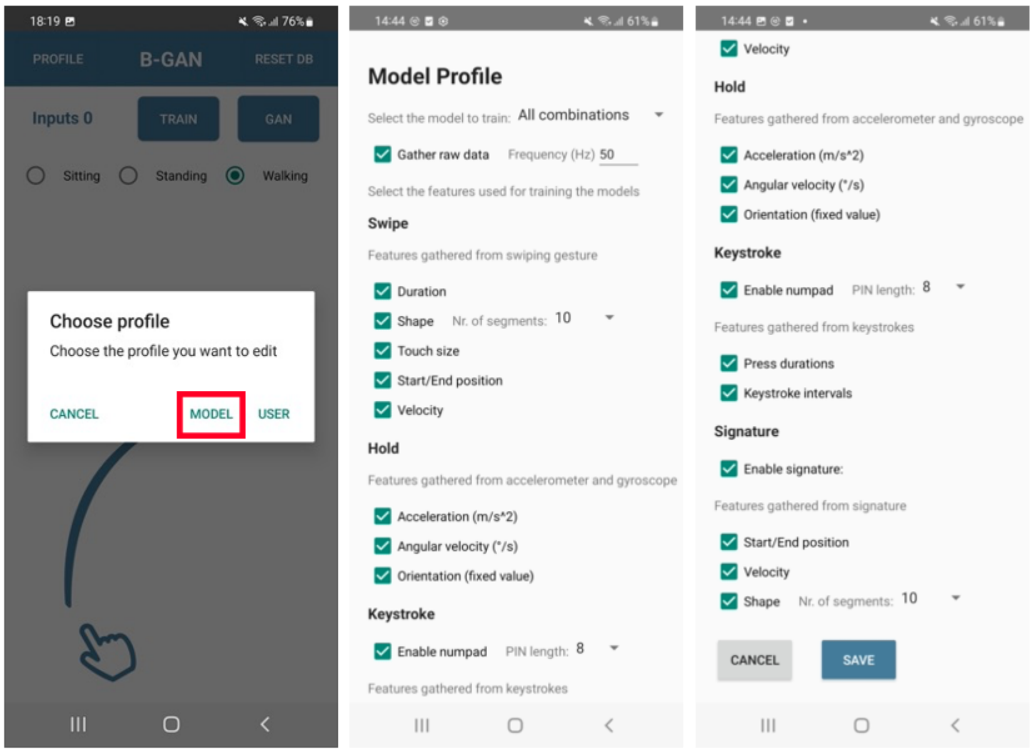
Finally, from within this view research can also toggle the raw data collection functionality (described in Chapter 4). In this context, B-GAN allows to specify the frequency (in Hz) at which this raw samples are collected during the data collection phase.

The model profile view is visually highlighted in Figure 3.7.

3.4 CSV files structure

After the conclusion of a given testing procedure, B-GAN offers the possibility to save the generated results as a set of `.csv` files. A given `.csv` file will contain the information related to a specific aspect of the data gathering or model testing phase.

In this context, the following files can be generated after the conclusion of a testing procedure:



Sitting Standing [✓ Walking]

Select the model to train: All combinations

✓ Gather raw data Frequency (Hz) 50

Select the features used for training the models

Swipe

Features gathered from swiping gesture

✓ Duration
✓ Shape Nr. of segments: 10
✓ Touch size
✓ Start/End position
✓ Velocity

Hold

Features gathered from accelerometer and gyroscope

✓ Acceleration (m/s^2)
✓ Angular velocity (°/s)
✓ Orientation (fixed value)

Keystroke

✓ Enable numpad PIN length: 8

Features gathered from keystrokes

✓ Press durations
✓ Keystroke intervals

Signature

✓ Enable signature:

Features gathered from signature

✓ Start/End position
✓ Velocity
✓ Shape Nr. of segments: 10

Figure 3.7: Model profile views

- The `featureData` file contains information on the features and models that were enabled during the training phase. It can be fundamentally seen as a summary of the model profile view's settings.
- The `rawData` file includes the raw samples collected (as a continuous stream and the specified frequency) throughout the training phase.
- The `realResults` file contains the training results (performance metrics) of the generated models, together with additional information related to the individual training times and number of samples used to build the models.
- The `realSwipes` file highlights the values of the individual features related to the train interactions gathered during the data collection phase.
- The `resourceData` file contains information on the resources used by the system during the training phase in terms of (min, max, avg) CPU frequency, (min, max, avg) memory usage and battery power draw.
- The `testAuthentication` file contains the authentication results and times for the individual test interaction across all the generated models.

- The `testResults` file contains the testing results (performance metrics) of the generated models, together with additional information related to the individual average testing times and number of samples used to test the models.
- The `testSwipes` file highlights the values of the individual features related to the test interactions gathered after building the models.
- The `userData` file contains the personal information provided for a given user.
- The `SUSData` file contains the respoonse to the SUS usability questions.

Additionally, if the training procedure involved the generation of synthetic samples, the following additional files are generated:

- The `ganResults` file contains the training results (performance metrics) of the models trained following the generation of the synthetic samples. This files is generated instead of the `realResults` one when the GAN training is performed.
- The `ganSwipes` file contains the values for the individual features related to the synthetic interactions generated by the adversarial network.
- The `realSwipesNormalized` file contains the interactions' values normalized (using min/-max scaling) for the purpose of training the adversarial network.

Chapter 4

Features & Related Models

4.1 Models description

Throughout the course of a single interaction, B-GAN makes use of a diverse set of features in order to correctly profile a given user. These features are collected using a variety of device sensors and can be divided into four different categories which ultimately make up the individual models that are part of the training procedure. Notice that, although these four models (Hold, Swipe, Keystroke & Sign) represent the smallest entities that are generated from the training process, B-GAN also allows to easily combine these individual units in order to highlight how the different models' combinations affect the final accuracy of the system.

In this section, we will provide a description of these four models together with their related features.

4.1.1 Hold

The HOLD model tries to capture the way in which the mobile device is held by the user throughout the course of a given interaction. Due to this fact, data related to the HOLD model is captured when performing *all* of the individual (Swipe, Keystroke & Signature) gestures that are part of a given interaction. The data used to build the HOLD model is gathered via three device sensors:

- **Accelerometer**: Returns the current acceleration along three axes (X, Y, Z) in $\frac{meters}{seconds^2}(\frac{m}{s^2})$.

- **Gyroscope**: Returns the angular velocity along three axes (X, Y, Z) in $\frac{radians}{seconds}(\frac{rad}{s})$.
- **Magnetometer**: Returns the currently perceived ambient magnetic field along the three axes (X, Y, Z) in Micro-Tesla (uT).

During each interaction, data points generated from these sensor are saved as continuous streams which are then used to compute the corresponding individual training values. Statistics generated from the output of the Accelerometer and Gyroscope are saved as-is and used during training. Additionally, the Accelerometer and Magnetometer sensor data is used in conjunction to compute fixed values related to the orientation (or angle of rotation) of the device along the three axes in $radians(rad)$ (see `getRotationMatrix()` and `getOrientation()` methods). These three axes are visually represented in Figure 4.1 (Image source) and can be defined as follows:

- The angle of rotation along the X axis (called **Pitch**) measures the angle between the device screen and the ground plane with values ranging from $-\frac{\pi}{2}$ to $\frac{\pi}{2}$.
- The angle of rotation along the Y axis (called **Roll**) measures the angle between a plane perpendicular to the device screen and a plane perpendicular to the ground with values ranging from $-\pi$ to π.
- The angle of rotation along the Z axis (called **Azimuth**) measures the angle between the device Y axis and the magnetic North Pole with values ranging from $-\pi$ to π.

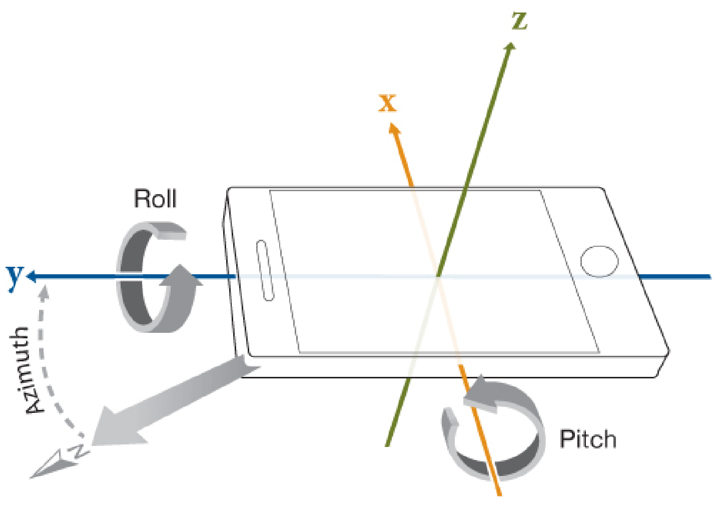


Figure 4.1: The three orientation axes

Each of the data streams related to acceleration, angular velocity and orientation and generated for a given interaction is summarised in the following metrics:

- Minimum and maximum X, Y and Z values.
- Average, Variance and standard deviation of the X, Y and Z values.

It is also important to specify that, during a given interaction, data points are gathered from the device sensors:

- When the user is *directly touching* the screen to perform a Swipe gesture.
- During the *whole duration* of the Keystroke gesture (after the first digit has been pressed).
- When the user is *directly touching* the screen to perform a Signature gesture.

4.1.2 Swipe

The SWIPE model tries to capture the way in which the user performs a swipe gesture on the device. In this context, a swipe is considered as a single uninterrupted gesture performed on the screen using a single point of contact. The data used to describe a swipe gesture and to build the SWIPE model is summarised as follows:

- Total **duration** of the gesture expressed in milliseconds.
- **Length** computed as the sum of euclidean distances between the set of individual points making up the swipe gesture (each point corresponding to a single call of the `onTouchEvent(MotionEvent.ACTION_MOVE)` and `handleSwipeEvent(MotionEvent.ACTION_MOVE)` methods performed during the swipe, see code documentation for more details).
- Minimum, maximum, average, initial and final size of the **screen area touched** by the single point of contact.
- **Start and end values** for the X and Y coordinates.
- Minimum, maximum, average, variance and standard deviation of **velocity** value for the X and Y coordinates (These values are obtained from the set of velocities measured between the individual points making up the swipe gesture).

In addition to these "base" features, a swipe segmentation technique was introduced in order to capture some information related to the shape of the provided gesture. In this context, the swipe gesture is divided into a predefined number of equally sized **segments** (this number can be freely changed in the model profile view). Each segment is identified by a start and end point. This means that a gesture with k segments will have $k+1$ points identifying them. For each of the generated swipe segments, the system measures what percentage of the total X and Y distances covered by the swipe gesture ($Dist_x$ and $Dist_y$ in the image below) are spanned by the segment at hand (e.g. A segment covering 10% of the total X distance and 7% of the total Y distance will be assigned the values $X_S = 0.10$ and $Y_S = 0.07$). Additionally, these percentages may have a positive or negative value depending on the relative position of the segment start and end points (i.e. left-to-right and bottom-to-top generate positive values, the inverse generates negative ones). The generated values help to gauge the shape of the swipe by capturing the X and Y displacement related to a given portion of the gesture. Naturally, incrementing the number of segments will increase the granularity of the gathered information. On the other hand, this will also increase the dimensionality of the resulting model which will treat each X and Y displacement value as a distinct feature. This technique is highlighted visually in Figure 4.2.

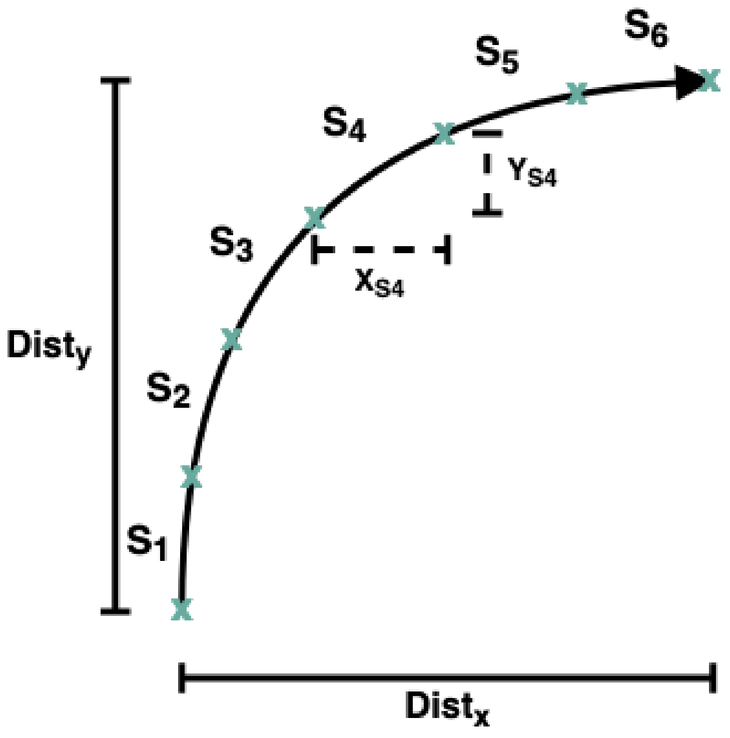


Figure 4.2: Swipe segments

4.1.3 Keystroke

The KEYSTROKE model tries to capture the way in which the user inserts a PIN of predefined length using a 10-digit numpad. In this context, a full keystroke gesture is recorded from the moment in which the user presses the first digit to the moment in which the last one is pressed with regards to the currently selected PIN length. The features used to build the KEYSTROKE model are taken from Buriro et al., 2015 [1] and are visually represented in 4.3. In this context, we have:

- The duration of the individual keystrokes making up the full PIN sequence (D1, D2, D3 and D4).
- The duration of the intervals between keystrokes (F1Type1, F2Type1, F3Type1).
- The duration of the intervals between the end of each keystroke (F1Type2, F2Type2, F3Type2).
- The duration of the intervals between the start of each keystroke (F1Type3, F2Type3, F3Type3).
- The full duration of the keystroke sequence (FType4).

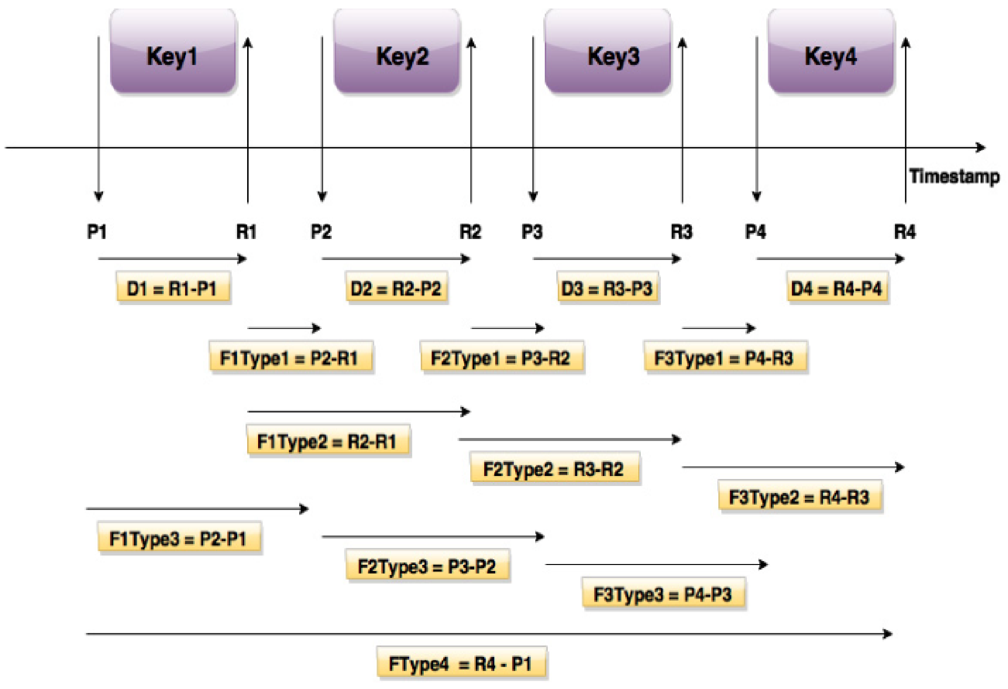


Figure 4.3: Keystroke features

4.1.4 Signature

The SIGNATURE model tries to capture the way in which the user inserts his/her signature on the mobile device. Unlike the swipe gesture, the signature does not necessarily require to be completed in a single uninterrupted motion. The feature of the SIGNATURE model are taken from Buriro et al., 2016 [2] and include:

- Start and end values for the X and Y coordinates.
- Euclidean distance between the start and end points.
- Standard deviation related to the set X and Y values associated with the points making up the signature.
- X and Y values related to the area spanned by the signature.
- Maximum and average **velocity** values for the X and Y coordinates (These values are obtained from the set of velocities measured between the individual points making up the signature gesture).

Additionally, the segmentation technique described for the Swipe gesture is also utilised for the signature model.

4.2 Combining the models

During the training phase, the previously described features can be freely combined to create models of arbitrary size. In this context, one way to create larger models (starting from the individual ones described at the beginning of this chapter) is to simply concatenate the collected features during the training procedure. This approach (which is visually highlighted in Figure 4.4) allows for the creation of individual models that should conceptually take into account information provided by different types of gestures. The drawback to this approach resides in the fact that adding additional features to a given model will increase its dimensionality and, consequently, the risk that the model will overfit the training data. Nonetheless, during the training procedure described in Chapter 5 the application is instructed to generate (using this approach) all possible model combinations (of any size) starting from the four individual ones. This decision leads to a more complete set of results (which are described and discussed in Chapter 6).

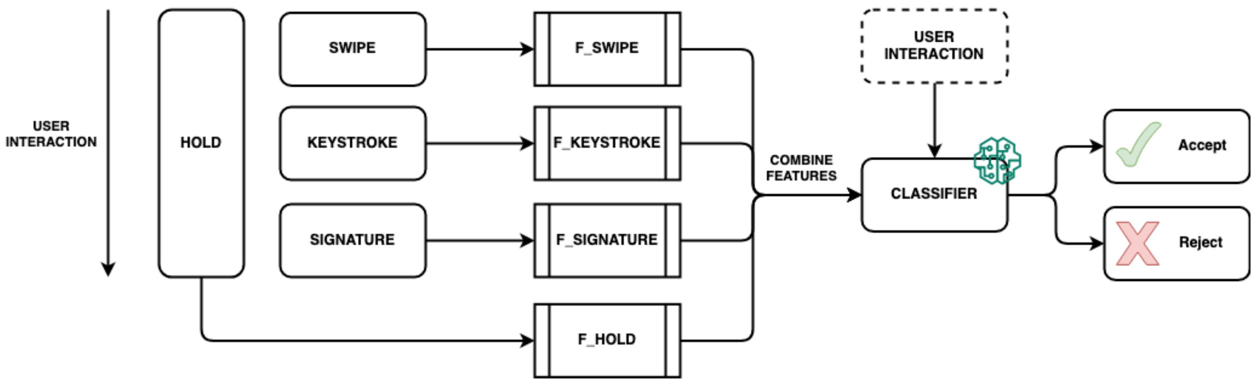


Figure 4.4: Feature concatenation approach

Another viable way to combine the data collected from the various gestures is to train the (four) individual models independently and to combine the individual predictions to generate a classification results for a given interaction. In this context, the individual models take part in a (weighted) voting procedure and the generated structure can be defined as a (weighted) voting ensemble (Figure 4.5).

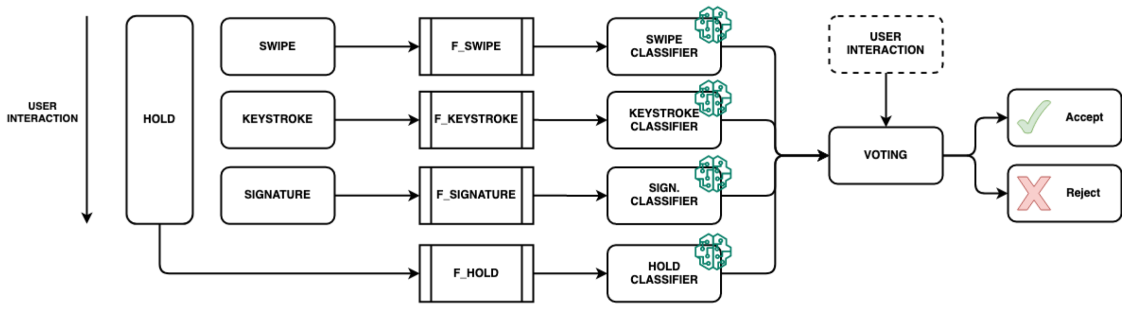


Figure 4.5: (Weighted) voting ensemble approach

The approach employed to compute the weight related to the vote of each model is taken from Section 4.4 of Buriro et al., 2016 [3]. In this context, a success index is computed from the error rate associated with each model after the training procedure. For a binary classifier, the error rate is expressed as:

$$er(c) = 1 - Accurary = 1 - \frac{TP + TN}{TP + FP + TN + FN} \tag{4.1}$$

Notice that, in the context of a one-class classifier, only positive samples are collected when training the models. Therefore, the error rate becomes:

$$er(c) = 1 - \frac{TP}{TP + FN} = 1 - TAR = FRR \tag{4.2}$$

In other words, the error rate is simply the false rejection rate expressed as a numerical value between 0 and 1. Starting from this error rate, the success index is expressed as:

$$index(c) = 1 - \frac{er(c)}{\sum_{i=1}^{n} er(i)} \tag{4.3}$$

The weight of a given model is then calculated using the following formula:

$$weight(c) = \frac{index(c)}{\sum_{i=1}^{n} index(i)} \tag{4.4}$$

4.3 OneClassClassifier

The models generated by the presented application are built using the `OneClassClassifier` class [7] which is made available as part of the `weka.classifiers` package [5]. The one-class classifier implementation offered by Weka is based on Hempstalk et al., 2008 [4]. The use of Weka classifiers in the context of an Android application is made possible by the WEKA Machine Learning framework as an Android Library [6].

In the context of this work, the trained classifiers are initialised with the following hyperparameters:

Option	**Value**
`-num`	`weka.classifiers.meta.generators.GaussianGenerator`
`-nom`	`weka.classifiers.meta.generators.NominalGenerator`
`-trr`	0.001
`-tcl`	1
`-cvr`	10
`-cvf`	10.0
`-P`	0.5
`-S`	1
`-W`	`weka.classifiers.trees.RandomForest` with 100 trees

Table 4.1: OneClassClassifier hyperparameters

4.4 Raw data collection

As mentioned in Section 3.3, B-GAN allows for the collection of raw training data, where information from the sensors is captured at regular intervals throughout the course of a given interaction. This is different from the traditional way in which training data is gathered where a full interaction will generate only one training sample. In this context, the number of entries generated in a single second is dictated by the frequency specified under the relevant setting in the model profile view (default is 50 Hz). When providing the training interactions, the `RawDataCollector` (described in Section 2.1) gathers data from the sensors only when the user is touching the screen (for the Swipe and Signature gestures) and from the first to the last key press (for the Keystroke gesture). In this regard, each sampling from the sensors will generate an entry containing information on: the current size of the point of touch on the screen, the X and Y coordinates and velocities, the X, Y and Z values for the accelerometer and gyroscope sensors and the X, Y and Z values for the orientation. Additionally, a gesture identifier is saved for each entry to indicate to which specific part of the interaction a given data points is referring to (1 = Swipe, 2 = Keystroke, 3 = Signature).

Notice that this part of the application was introduced to lay the foundation for potential future work. At the moment, the collected raw data can be saved in the form of a dedicated `.csv` file but is not used to train any of the presented models.

Chapter 5

Structure of the Training and Testing Procedure

In order to come up with a set of tangible results related to the accuracy of the proposed system, a specific workflow was laid out in order to highlight the details of the training and testing procedures. In this context, it was decided to collect data from a set of users that would provide the interactions required to train (and subsequently test) the generated models. In this chapter we will generally define the procedure that was carried out with each individual user in order to to achieve this goal. Additionally, we will discuss how the four main performance metrics related to the B-GAN system were generated. Finally, we will provide a brief overview of the kind of users that were involved in the data gathering phase.

5.1 Initial performance metrics

For each of the interested users, this data gathering phase was fundamentally organised into two main steps. After an initial introduction (where the fundamentals of the application were clearly shown and explained) the given user was asked to provide an initial set of training interactions. In this context, the application was setup to have all gestures and fundamental features enabled (this included the raw data collection process executed at a frequency of 50 Hz). The keystroke gesture was set to posses a total of 8 digits and the swipe and signature gestures were divided into ten segments when processed. Each user was asked to provide at least 30 training interactions. Additionally, it was asked for such training interaction to be provided using three different body postures (10 interactions per body posture). Upon completion of this initial step, the training procedure was started by the person supervising the experiment. In this context, the application was setup to train all possible models' combinations. This first data gathering and training step is visually highlighted in Figure 5.1.

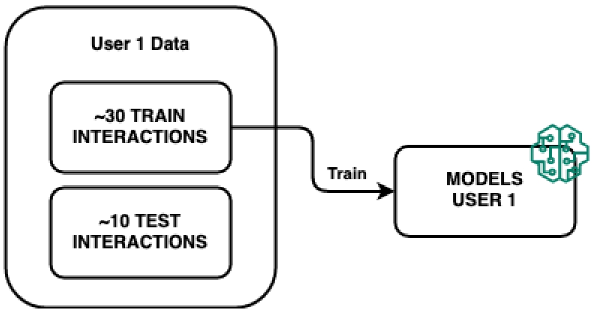


Figure 5.1: Training phase

Once the training procedure was completed by the application, the mobile device was given back to the end user which was asked to provide another set of at least 10 interactions. In this case, the interaction could be provided in any of the three body postures used during training. Once this second step was completed, the device was handed back to the researcher which would trigger the generation of the test results and save them as an initial set of .csv files. This second step is shown in figure 5.2.

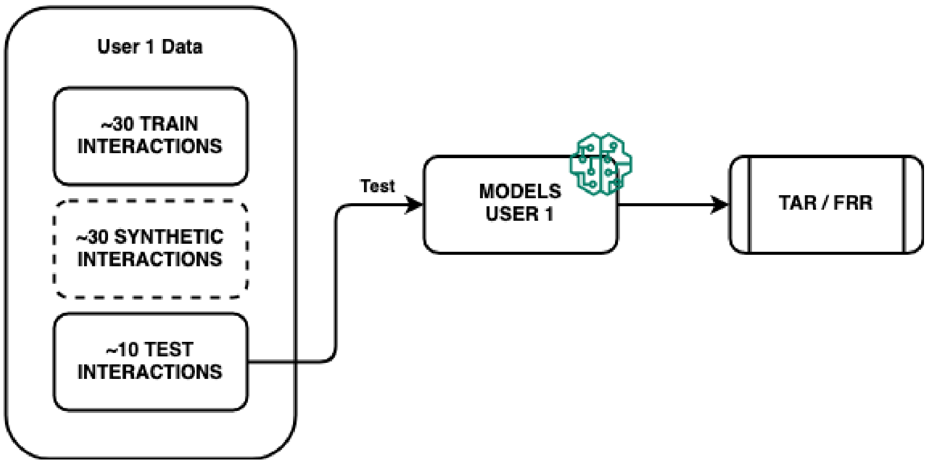


Figure 5.2: Testing phase

Upon completion of this second step, the user was informed that no more interactions would be required to be registered via the application. At this point, the researcher was given the phone back and a second training phase involving the adversarial network and the generated synthetic samples was started. In this context, the set of 30 original training interactions were used to train the adversarial network which would subsequently generate an equivalently sized set of synthetic samples that would take part in the subsequent training procedure. This second training phase is shown in Figure 5.3.

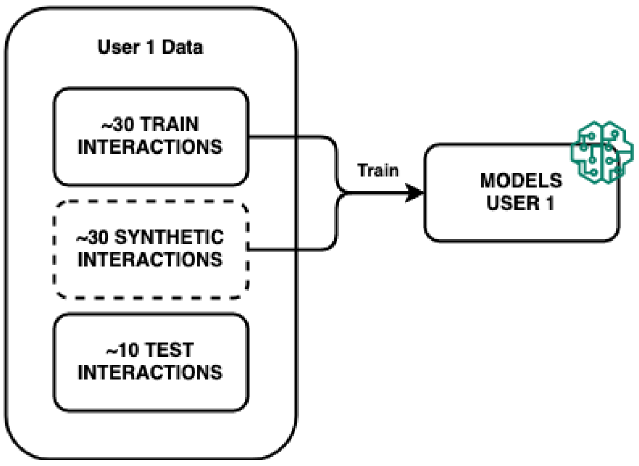


Figure 5.3: Training phase with synthetic samples

The second testing procedure was carried out using the 10 testing samples previously collected from the user. At this point, the set of generated performance metrics, stemming from both training and testing phases, included only the values for the True Acceptance Rate (TAR) and False Rejection Rate (FRR). This was due to the fact that these are the only two metrics that can be computed starting from a collection of positive samples. In this context, to produce the missing metrics, the generated models need to be "attacked" by non-genuine interactions. For this purpose, an "Attack" script was written (the description of which can be found in Section 5.2).

Additionally, a step-by-step guideline document (titled `Data collection guidelines.docx`) further highlighting (with screenshots) the process explained in this section has been attached to this report submission.

5.2 The Attack Script

In order to automatically execute the calculation of the True Rejection Rate (TRR) and False Acceptance Rate (FAR) for all users (and corresponding models) an “Attack” script was written. This simple script is designed to be executed upon full completion of the data gathering phase (across all the interested users). In short, the script executes (for each end user) the training step described in the previous section. This means that, for each user, the models are built using the set of genuine training samples originally provided. Note that the same seed is used at training time, effectively resulting in the same models generated when the original data was collected. Upon completion of this (identical) training phase, the models related to a given user are tested against all the sets of 40 training + testing samples belonging to the datesets related to each of the other users. In this context, the other users act as attackers that the models should ideally identify and reject. This additional testing phase, where the models are put against non-genuine interaction, is required to generate the missing TRR and FAR metrics and is visually highlighted in Figure 5.4. The process is repeated for both traditional and GAN-trained models.

In order to be successfully executed, the script requires a `data` folder containing, for each user, the `.csv` files related to the train, GAN and test interactions. An example of the set of files required for a single user looks as follows:

```
...
2022-10-26_14-20_ganSwipes.csv
2022-10-26_14-20_realSwipes.csv
2022-10-26_14-20_testSwipes.csv
...
```

A simplified version of the logic governing the execution of the Attack script is highlighted in the following pseudocode snippet:

```
initialise realResults.csv
initialise ganResults.csv

get users_list
(where each user is identified by one file with suffix _realSwipes.csv)

for model_type in ['nonGAN', 'GAN']:
    for user in users_list:
        train_interactions = []
        test_interactions = []

        add train_interactions from user_realSwipes.csv
        if model_type == 'GAN':
            add train_interactions from user_ganSwipes.csv
        train models using train_interactions

        for attacker in users_list:
            if attacker == user:
                continue
            add test_interactions from attacker_realSwipes.csv
            add test_interactions from attacker_testSwipes.csv

        for test_record in test_interactions:
            for model in models:
                run test_record against model

        if model_type == 'GAN':
            add test_interactions results to ganResults.csv
        else:
            add test_interactions results to realResults.csv
```

Listing 5.1: Attacker script logic

5.3 The Data Gathering Process

Using the methodology described in the two previous sections, data was collected from a total of twenty users. These users represent a varied sample of individuals with different backgrounds and affinities with technology. In this context, twelve users were males and eight were females. In terms of age groups, the majority of the interested candidates (ten out of twenty) were in the middle group belonging to the age range 21-40. Additionally, four users belonged to the range 41-60 and another four to the one of 60+. Only one candidate belonged to the age group 0-20 and another candidate decided not to disclose its age when providing the interactions. Throughout the course of the experiment, most of the users (twelve out of twenty) decided to hold the mobile device with the right hand, whilst seven used the left one. Only one user used both hands during the experiment.

In the next chapter, the results obtained with the data gathered from the described cohort of users will be highlighted using the previously mentioned performance metrics.

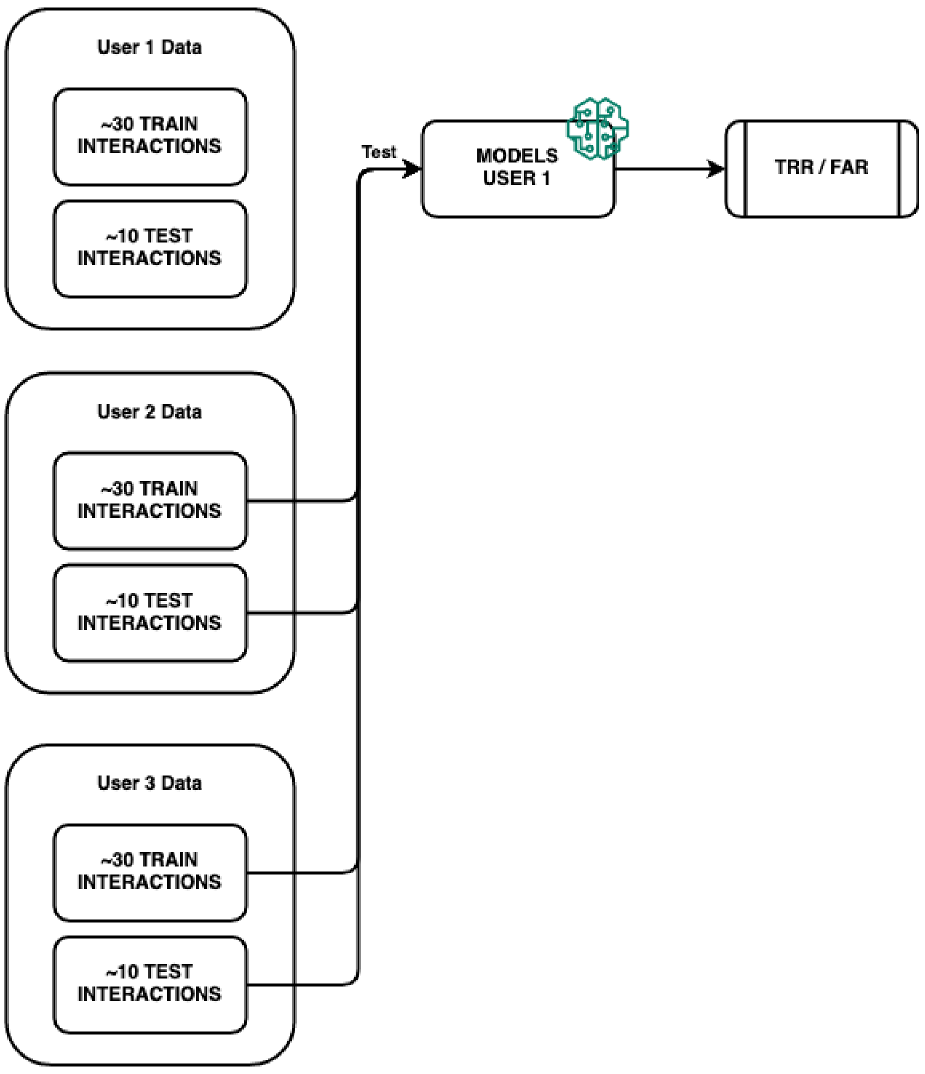


Figure 5.4: Testing phase using the ”Attack” script

Chapter 6

Results

Using the data generated from the procedures discussed in the previous chapter, a set of results describing the various characteristics of the proposed system are generated and visualised. This results not only describe the accuracy of the generated models (expressed using the already mentioned performance metrics), but also cover additional details related to training and classification times and indicators related to the mobile device resources consumed during the training process. In this chapter, we will provide a brief overview of the most relevant result values and we will discuss what such result imply in the context of the presented application. Notice that the presented values are computed by averaging the results obtained across the different users.

6.1 Result values

6.1.1 Non-GAN results

In the following subsection, the performance metrics computed from the models generated using solely the genuine samples are presented. Figure 6.1 highlights the results obtained from all generated models combinations. The bar-chart situated at the top of the image highlights the values related to the True Acceptance Rate (indicated in green) and False Rejection Rate (indicated in red). On the other hand, the bottom chart similarly displays the values related to the True Rejection Rate and False Acceptance Rate. The corresponding model name can be observed on the bottom x-axis. The first four bar groups refer to the individual (Hold, Swipe, Keystroke and Signature) models, `FULL` refers to the single model trained on all gathered feature and `WEIGHTED_ENSEMBLE` refers to the results obtained from the voting procedure executed among the individual models.

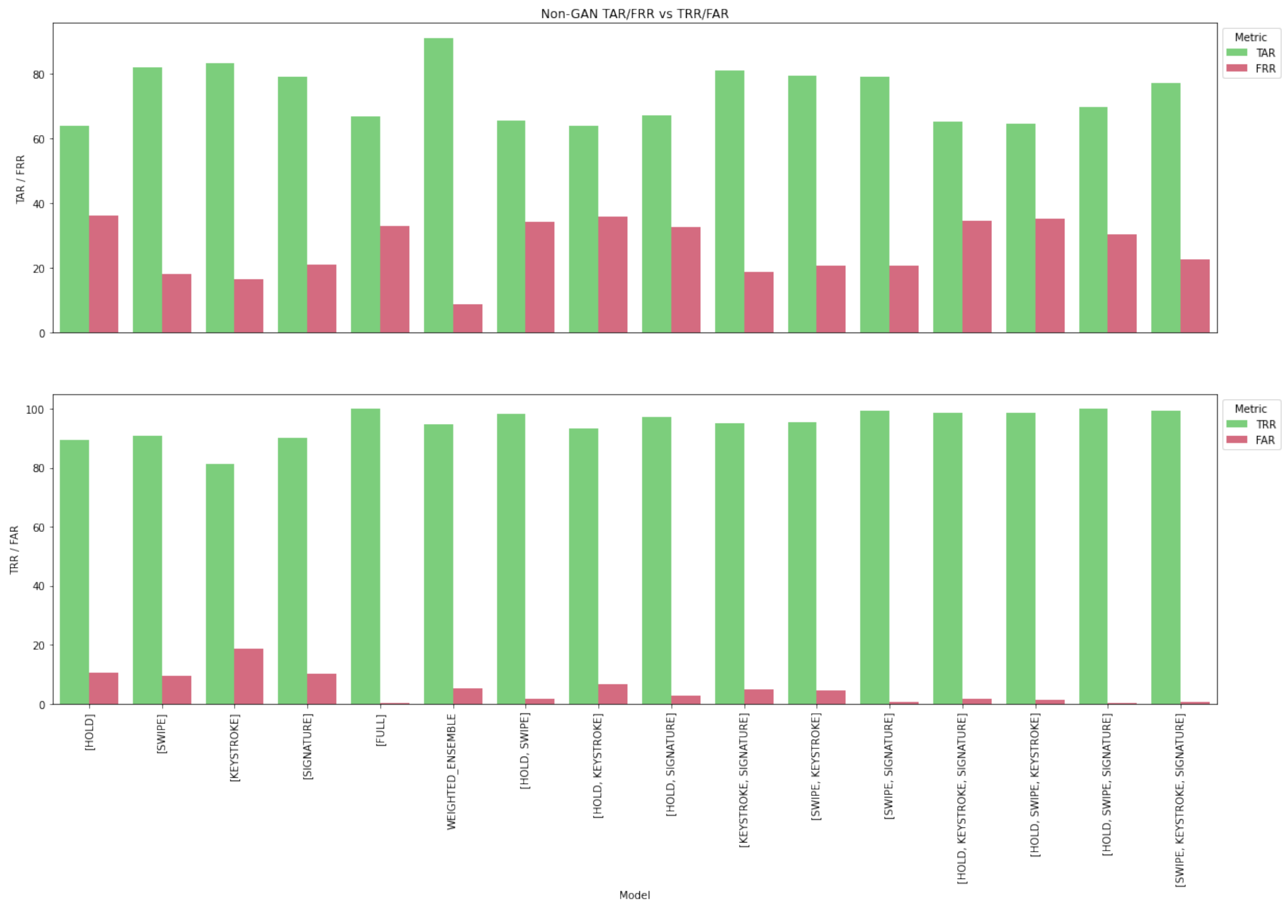


Figure 6.1: Performance metrics for all models (non-GAN)

Detailed values related to the TAR / FRR / TRR / FAR for the four individual models can be observed in the confusion matrices highlighted in Figure 6.2.

Finally, the confusion matrices for the full model and for the weighted ensemble can be found in Figure 6.3.

6.1.2 GAN results

In a similar fashion, the individual results obtained from training the models using the synthetics samples (in conjunction with the genuine ones) are shown in Figure 6.4.

Once again, the most relevant confusion matrices are found in Figure 6.2 and Figure 6.6.

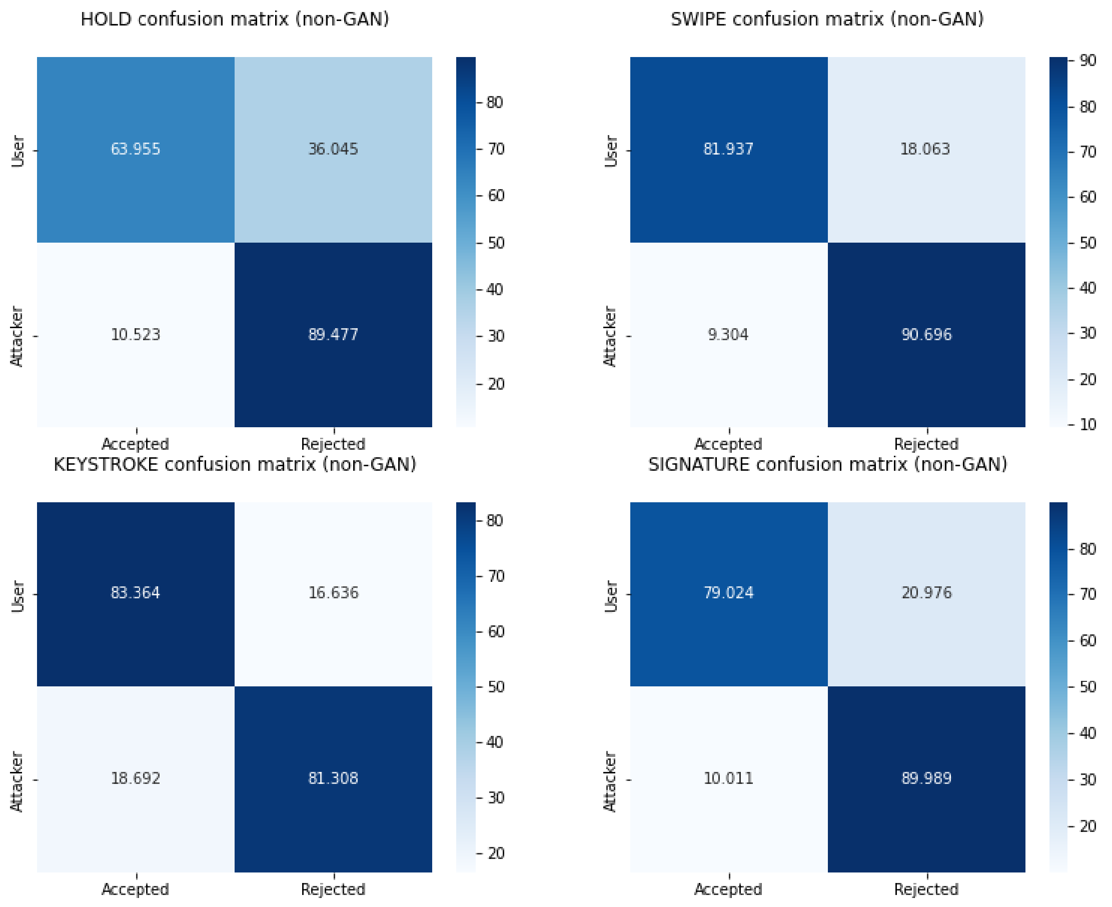
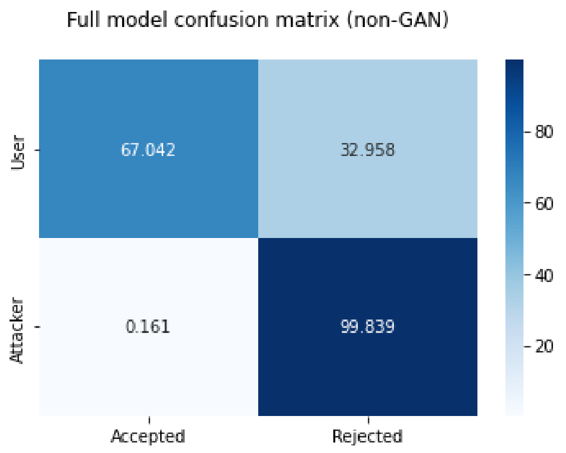
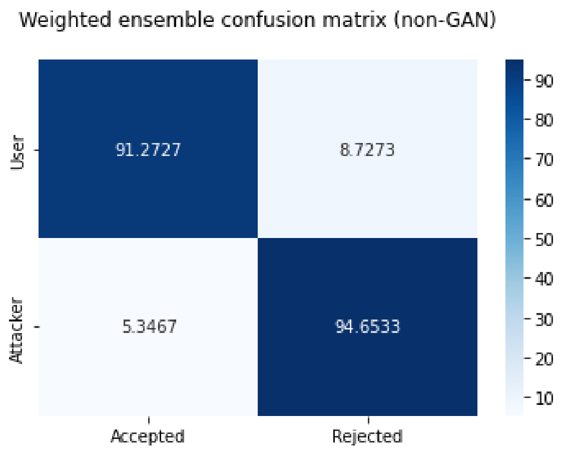


Figure 6.2: Confusion matrices for the individual models (non-GAN)



(a) Full model



(b) Weighted ensemble

Figure 6.3: Confusion matrices for full model and ensemble (non-GAN)

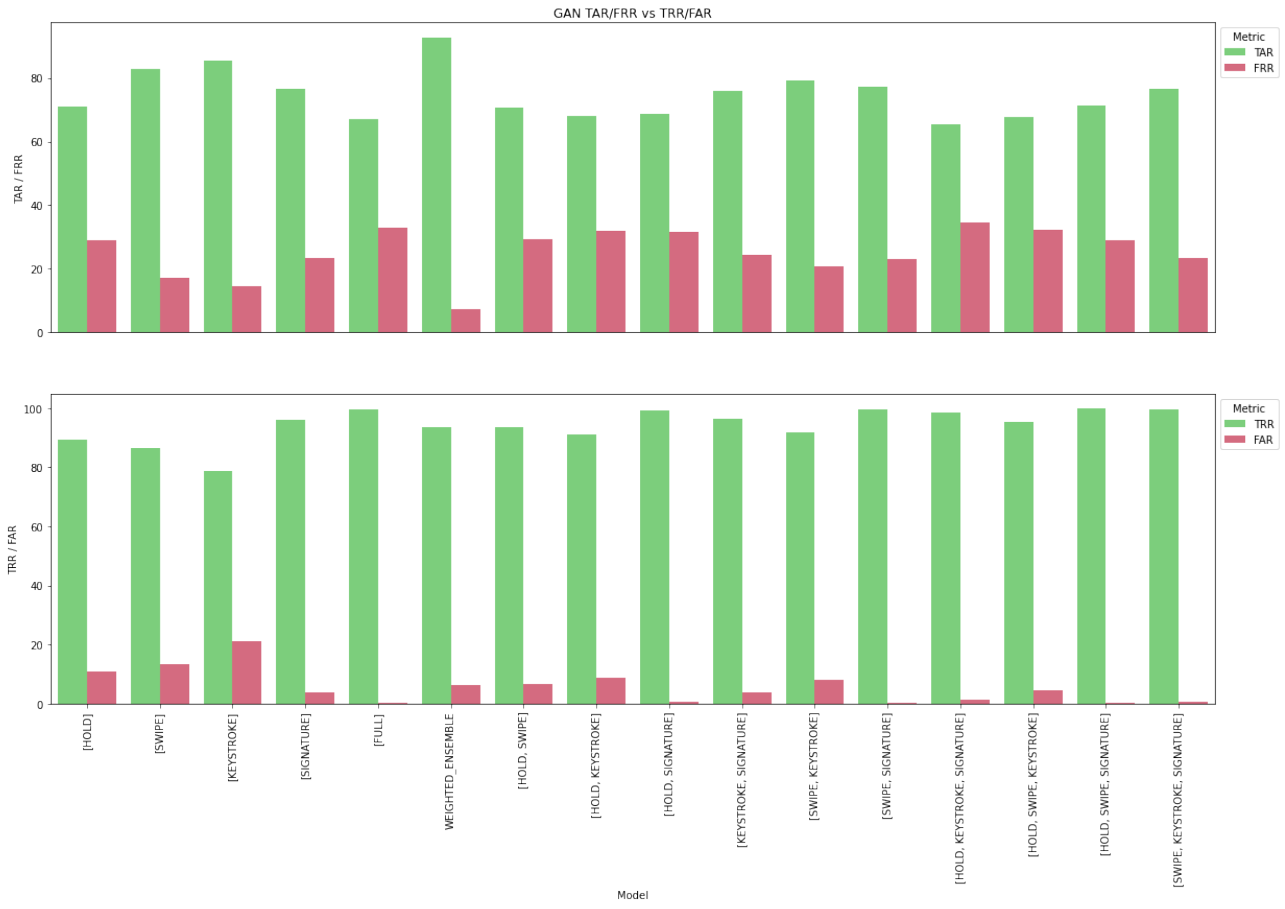


Figure 6.4: Performance metrics for all models (GAN)

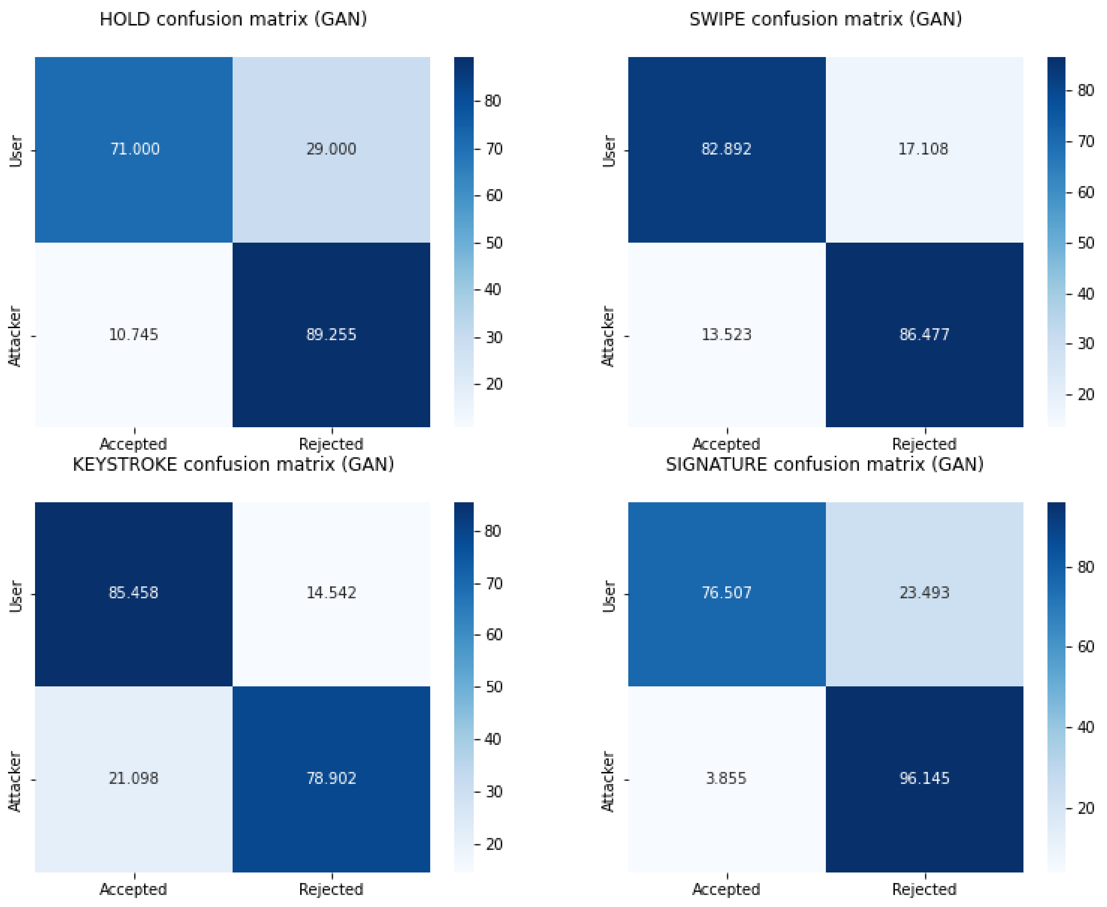
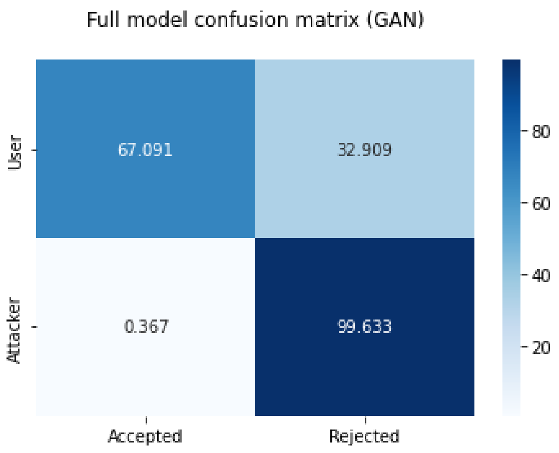
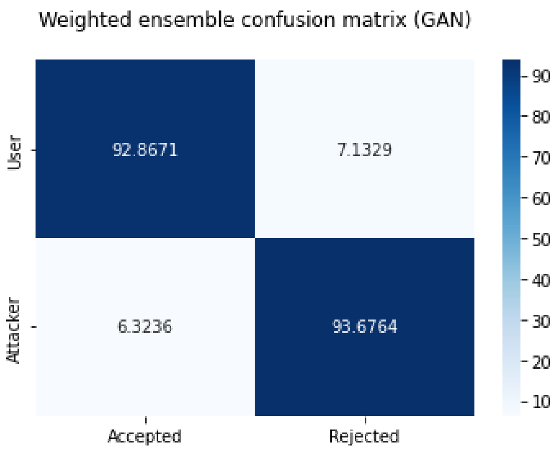


Figure 6.5: Confusion matrices for the individual models (GAN)



(a) Full model



(b) Weighted ensemble

Figure 6.6: Confusion matrices for full model and ensemble (GAN)

6.2 Discussion

In this section, a few observation related to the presented results are made. Firstly, it is possible to observe how, in a general sense, the system rarely accepts interactions provided by non-genuine users. The low values related to the false acceptance rates are observed fundamentally across all trained models. On the other hand, it is clear that the system does not perform equally as well when interacting with genuine users, were results related to the false rejection rate tend to be higher than the FAR counterpart. When observing the metrics tied to the individual models, it can be noticed how the ones generated from the three gestures making up an individual interaction produced generally satisfactory results (with TAR and TRR values ranging between 79-90%). On the other hand, the models generated from the users' holding behavior did not reveal themselves to be as accurate as the touch-based counterparts. In this regard, we can observe TAR values of 63% and 71% (for the non-GAN and GAN version respectively). The strategy that by far yielded the best results has been the voting procedure conducted among the four individual models. In this context, the weighted ensemble produced TAR and TRR values of 91.3% and 94.7% respectively (92.9% and 93.7% for the GAN version). Because of this reason, this strategy is also chosen when displaying the visual feedback related to the acceptance (or rejection) of a given test interaction when utilising the application. On the other hand, the full model generated from the concatenation of all collected features yielded fairly poor results with regards to the TAR (whilst retaining very high values for the TRR). In other words, the model was very conservative when classifying a given interaction. This was an expected behavior brought upon by overfitting caused by the high dimensionality related to the large number of features on which the model was trained. These previous observations are generally applicable to both the gan and non-gan versions of the proposed models. One general difference observable across most models built using the synthetic samples is that they tend to be slightly more "permissive" when it comes to evaluating a given interaction as genuine. Naturally, this fact yielded to generally higher TAR values (but also to slightly lover TRRs).

Chapter 7

Future work & Appendix

In terms of future developments for this project, a few avenues are highlighted as potential extensions of the presented concept. Firstly, as introduced in Section 4.4, the application currently allows for the collection of raw data in the form of a continuous stream of information from the mobile device. This data is currently not used during the training process. Conceptually, the information carried over by these raw samples are fundamentally different from the ones that are currently used to build the presented models. In this regard, training a set of model starting from this data and potentially comparing their performance to the existing solution could be an interesting direction to extend the currently offered functionalities.

Secondly, a more extensive work related to feature engineering, feature selection and hyperparameter tuning could be carried out in order to further reduce the dimensionality and improve the overall performance of the existing models.

Finally, it must be noted how a wide array of machine learning algorithms could potentially be used to tackle the problem at hand. Testing different underlying algorithms and comparing their performance (using the already collected data) to the currently offered solution could also be a viable approach to further extending this work.

7.1 Appendix

In this appendix, some of the secondary changes and fixes made to the original application (SwipeGAN) are highlighted:

- The calculation of the normalised values used to train the adversarial network produced results outside the expected 0-1 range. This was caused by the fact that hard-coded maximum values were used in the calculation for each existing feature. This error was fixed in B-GAN.
- Originally, the result returned for the velocity-related features was direction dependent. This means that, depending on the direction of a given gesture, negative values could be produced. In B-GAN, the velocity calculation ignores the gesture's direction.
- The user profile was converted into a proper view (from the original PopupWindow-based implementation).
- The existing code in the `Swipe` and `DBHelper` classes was refactored in order to address the instances of redundant code that were found when inspecting them.

References

[1] Attaullah Buriro, Bruno Crispo, Filippo Del Frari, and Konrad Wrona. "Touchstroke: Smartphone User Authentication Based on Touch-Typing Biometrics". In: *New Trends in Image Analysis and Processing – ICIAP 2015 Workshops.* Ed. by Vittorio Murino, Enrico Puppo, Diego Sona, Marco Cristani, and Carlo Sansone. Cham: Springer International Publishing, 2015, pp. 27–34. ISBN: 978-3-319-23222-5.

[2] Attaullah Buriro, Bruno Crispo, Filippo Delfrari, and Konrad Wrona. "Hold and Sign: A Novel Behavioral Biometrics for Smartphone User Authentication". In: *2016 IEEE Security and Privacy Workshops (SPW).* 2016, pp. 276–285. DOI: `10.1109/SPW.2016.20`.

[3] Attaullah Buriro, Bruno Crispo, Filippo Del Frari, Jeffrey Klardie, and Konrad Wrona. "ITSME: Multi-modal and Unobtrusive Behavioural User Authentication for Smartphones". In: *Technology and Practice of Passwords.* Springer International Publishing, 2016, pp. 45–61. DOI: `10.1007/978-3-319-29938-9_4`. URL: `https://doi.org/10.1007%2F978-3-319-29938-9_4`.

[4] Kathryn Hempstalk, Eibe Frank, and Ian H. Witten. "One-Class Classification by Combining Density and Class Probability Estimation". In: *Machine Learning and Knowledge Discovery in Databases.* Ed. by Walter Daelemans, Bart Goethals, and Katharina Morik. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 505–519. ISBN: 978-3-540-87479-9.

[5] *Weka classifiers package.* `https://weka.sourceforge.io/doc.dev/weka/classifiers/package-summary.html`.

[6] *WEKA Machine Learning framework as an Android Library.* `https://github.com/andrecamara/weka-android`.

[7] *Weka OneClassClassifier.* `https://weka.sourceforge.io/doc.packages/oneClassClassifier/weka/classifiers/meta/OneClassClassifier.html`.