Date of acceptance Grade

Instructor

Testing for Convolutional Neural Network-based Gait Authentication in smartphones

Anh, Ta Tuan

Helsinki April 29, 2020

UNIVERSITY OF HELSINKI
Department of Computer Science

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Laitos — Institution — Department
Faculty of Science	Department of Computer Science

Tekijä — Författare — Author
Anh, Ta Tuan

Työn nimi — Arbetets titel — Title

Testing for Convolutional Neural Network-based Gait Authentication in smartphones

Oppiaine — Läroämne — Subject
Computer Science

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Master's programme	April 29, 2020	45 pages +

Tiivistelmä — Referat — Abstract

Most online fraud involves identity thief, especially in financial services such as banking, commercial services, or home security. Passwords have always been one of the most reliable and common way to protect user identities. However, passwords can be guessed or breached. Biometric authentications have emerged to be a compliment way to improve the security. Nevertheless, biometric factors such as fingerprint or face recognition can also be spoofed. Additionally, those factors require either user interaction (touch to unlock) or additional hardware (surveillance camera). Therefore, the next level of security with lower risk of attack and less user friction is essentially needed. gait authentication is one of the viable solutions since gait is the signature of the way humans walk, and the analysis can be done passively without any user interactions. Several breakthroughs in terms of model accuracy and efficiency were reported across several state-of-the-art papers. For example, *DeepSense* reported the accuracy of 0.942 ± 0.032 in Human Activity Recognition and 0.997 ± 0.001 in User Identification.

Although there have been research focusing on gait-analysis recently, there has not been a standardized way to define proper testing workflow and techniques that are required to ensure the correctness and efficiency of gait application system, especially when it is done in production scale. This thesis will present a general workflow of Machine Learning (ML) system testing in gait authentication using V-model, as well as identifying the areas and components that requires testing, including data testing and performance testing in each ML-related components. This thesis will also suggest some adversarial cases that the model can fail to predict. Traditional testing technique such as differential testing will also be introduced as a testing candidate for gait segmentation. In addition, several metrics and testing ideas will also be suggested and experimented. At last, some interesting findings will be reported in the experimental results section, and some areas for further future work will also be mentioned.

ACM Computing Classification System (CCS):
General and reference Document types Surveys and overviews
Applied computing Document management and text processing Document management Text editing

Avainsanat — Nyckelord — Keywords
Biometric Identification, Gait Classification, ML testing, Software testing

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — övriga uppgifter — Additional information

Contents

1 Introduction **3**

2 Gait authentication **4**

2.1 Authentication, biometrics, and gait 4

2.2 Smartphone-based gait authentication using deep learning 6

2.2.1 Gait segmentation 8

2.2.2 Gait classification 10

2.3 Gait authentication productionization 11

3 Traditional system testing vs ML-based system testing **12**

3.1 Traditional software testing and its limitations when applying to an ML system 12

3.2 Current ML testing techniques 13

3.2.1 Bug detection in data 14

3.2.2 Bug detection in ML models 14

3.2.3 Bug detection in the framework 16

4 Proposed testing solution **16**

4.1 Organization of related work and workflow in ML testing 16

4.1.1 V-model in software testing 17

4.1.2 Where, What, How to test? 17

4.2 Gait segmentation testing 19

4.2.1 Data testing 19

4.2.2 Extraction program testing 20

4.3 Gait classification testing 21

4.3.1 Data testing 22

4.3.2 Learning program testing 23

4.3.3 Model training infrastructure testing 24

4.3.4 Trained models testing 24

5 Experimental materials and procedure **25**

5.1 IDNet gait dataset 25

5.2 Experimental procedure 26

6 Experimental results **26**

6.1 Gait segmentation testing 26

6.1.1 Data testing 26

6.1.2 Extraction program testing 27

6.2 Gait classification testing 30

6.2.1 Data testing 30

6.2.2 Trained models testing 33

7 Discussion **37**

8 Conclusion **40**

References **40**

Acronyms

ROC - AUC Receiver operating characteristic - Area Under Curve. 25, 33–35, 38

CNN Convolutional Neural Network. 8, 10–12, 40

CPU Central Processing Unit. 25, 35

DL Deep Learning. 3, 6, 8, 10, 15, 16, 21, 24, 39, 40

FS Floor Sensor. 5

GPU Graphics Processing Unit. 35

ML Machine Learning. i, ii, 3, 4, 6, 8, 10–19, 24, 25, 37, 40

MV Machine Vision. 5

PIN Personal Identification Number. 4

RNN Recurrent Neural Network. 8

ROC Receiver operating characteristic. 1, 25, 33–36, 38

TFX Tensorflow Extended. 14

WS Wearable Sensor. 5

Glossary

False Positive Rate (FPR) False Positive Rate is one of the metrics that are commonly used for performance evaluation of binary classification problems. It tells the proportion of the negative test data that was mis-predicted as positive labels. Intuitively, the closer the value to 0.0, the better the classifier is. 34, 36, 38

False Rejection Rate (FRR) False Rejection Rate is one of the metrics that are commonly used in biometric authentication. It tells the proportion of the negative (a.k.a attacker) data that was correctly predicted as attacker. Put it simply, it is equivalent to the True Negative Rate in binary classification evaluation. Also, it is the compliment value of False Positive Rate (True Negative Rate + False Positive Rate= 1.0). Intuitively, the closer the value to 1.0, the better the classifier is. 4, 24, 25, 33–35, 38

True Acceptance Rate (TAR) True Acceptance Rate is one of the metrics that are commonly used in biometric authentication. It is equivalent to True Positive Rate that is used in binary classification evaluation. Intuitively, the closer the value to 1.0, the better the classifier is. 3, 24, 25, 33–35, 38

True Positive Rate (TPR) True Positive Rate is one of the metrics that are commonly used for performance evaluation of binary classification problems. It tells the proportion of the positive test data that was correctly predicted as positive labels. Intuitively, the closer the value to 1.0, the better the classifier is. 24, 34, 36, 38

1 Introduction

Gait authentication has emerged to be one of the new research directions for passive biometric authentication. Together with the development of Deep Learning (DL), gait authentication using DL models have become one of the main focuses in gait-analysis. Some promising experimental results have been reported [1, 2, 3]. As a result, productionizing the solution is a desirable next step. However, compared to research and development, building a software product requires a more solid quality assurance and more exhaustive testing to maintain the reliability of the product. Therefore, there is a strong need in having a proper way to test and verify the software system before deploying it to the customers.

Traditionally, testing a software system requires different levels of testing such as unit testing, component testing or integration testing. And the typical testing criteria is code coverage (line coverage, branch coverage, etc.). Nevertheless, compared to a traditional software system where there is a clear specification of the expectation of how the system should perform, an ML system cannot have a definite expectation. For example, if an ML product specification indicate that the system should be able to detect a picture of a cat as cat, the system cannot assure to return a correct prediction in every test. Specifically, an ML system contains the components in which the decision is made by using statistical models. And since the models are non-deterministic, the expected answers cannot always be 100% correct. As a result, simply using a small set of test data combined with the simple pass/fail assertion cannot evaluate the correctness of the system. And even if the system fails to predict in some particular samples, it does not always mean the system is not functional. Moreover, compared to traditional software systems, an ML system introduces additional testing space such as input data or model training infrastructure. They are the core components which drive the final results of the system. Therefore, additional testing is required in order to detect and prevent bugs from these components. As a result, the testing space in an ML system is also much wider than in a traditional software system. Therefore, applying the traditional testing workflow and testing criteria into an ML system is not comprehensive enough.

There have been some research focuses on testing ML applications, including white-box testing DL models. Most of the testing focuses are within computer vision domains [4, 5, 6]. Nevertheless, there has not been a standardized way of testing an end-to-end ML system, especially on gait authentication domain. Moreover, because the nature of input data in smartphone-based gait analysis is quite different from other mainstream domains such as computer vision, some of the testing techniques such as metamorphic testing introduced in computer vision are not directly applicable [4]. And thus, this thesis is going to propose a testing solution for gait authentication system by combining V-model [7] in traditional software testing with other testing techniques for testing non-deterministic software systems.

Specifically, this thesis leverages the well-known testing workflow (V-model) and some testing techniques (e.g. differential testing) in software testing, and combines them with some standard biometric authentication evaluation metrics (True Accep-

tance Rate (TAR), False Rejection Rate (FRR)) to provide a more well-thought testing solution in gait authentication domain. Also, this thesis will introduce a data testing solution for gait data by leveraging the unit testing methodology. In addition, this thesis will identify the testing space in gait authentication production pipeline in which the bugs might occur. The identification of testing space and the proposing testing solution mentioned in this thesis is domain dependent. Compared to other domains such as computer vision or image processing, gait analysis does not attract the same amount of interest. Therefore, performing tests in this application domain is an interesting but challenging topic. Within the scope of this thesis, some interesting testing ideas such as performing white-box testing in gait model, or advanced model debugging will not be covered. They will only be mentioned as possible research directions for future work.

The remaining structure of the thesis is organized by giving some background information about authentication, biometric authentication, and human-gait in section 2.1. After that, section 2.3 will give a high-level overview about smartphone-based gait authentication production pipeline. Subsequently, the review of current state-of-the-art of testing activities on ML applications and the organization of related workflow in ML testing for gait authentication will be introduced in section 3 and 4, respectively. In order to answer the questions *"Where, What, and How to test?"*, section 4 will present a proposal for testing activities on different validation phases (from V-model) such as unit testing, component testing, system integration testing, and user acceptance testing on the core components in gait authentication system. At last, the experimental results of the proposal in section 4 and some conclusions including areas for future work will be discussed in section 6 and 7, respectively.

2 Gait authentication

2.1 Authentication, biometrics, and gait

This section aims to give a high-level introduction about authentication, different ways of doing biometrics authentication, and how gait analyses are being used as a biometric authentication factor.

Firstly, authentication can be seen as a way to identify user's identity before granting permissions for him or her to get access to some sensitive information. For smartphone-based authentication, there are several approaches that are being used currently such as PIN/password, graphical-based methods, or biometrics [8]. Out of the three methods, PIN/password and graphical-based methods such as pattern are knowledge-based authentication. Although they are the most common ways of identity verification, there are some limitations with this approach: the required secret should be secure and easy-to-remember [8]. As a result, the secrets are usually re-used in multiple applications and the knowledge is closely related to the user (the places they have been to, their pet names or their date of birth) [8]. Unfortunately, with the current presence of social networks, this information is not difficult to ac-

quire, and therefore, the search space for an attacker is decreased and it makes this approach become more vulnerable [9].

Unlike knowledge-based authentication, biometrics authentication is another category of verifying user's identity by learning from the user's behavioral or physiological characteristics such as face, fingerprint, voice, iris, and gait that are distinctive to a person. If knowledge-based authentication is considered as "something you know", biometrics authentication can be categorized into two different categories: *behavioral biometric* which is "something you are" and *physiological biometric* which is "something you do" [3]. Specifically, fingerprint, face, and iris are examples of physiological biometrics, while behavioral biometrics cover human activity factors such as handwriting, gait, signature, and voice. Both categories are being used for identification purposes. However, the use of behavioural characteristics as personal trails is relatively new and diversified field of research [10]. Although behavioral biometrics are more volatile to changes over the long term (for example, the way of walking, writing or the keystroke dynamics may be affected by either physical or emotional status), they have the advantages of being non-intrusive and are more difficult to forge and spoof. Also, they can also be used together with other physiological biometric factors to improve recognition accuracy [11].

Among behavioral biometrics, gait is increasing its popularity in enhancing user identification [11]. By definition, gait represents the pattern of the way how a person walks [12]. Figure 1 illustrates the configuration of a gait period [13]. As can be seen in Figure 1, each gait period is divided into 8 different configurations and 2 different phases. An entire gait period can also be called a gait cycle (or a walking cycle), which consists of two steps: left step and right step (or vice versa), each is composed of 4 consecutive configurations. Visual pattern, dynamic weight/pressure distribution, and acceleration patterns are generated by the gait dynamics. At the moment, those observations and characteristic are being used to differentiate between different users. In gait recognition, there are three main different approaches: Machine Vision (MV)-based , Floor Sensor (FS)-based, and Wearable Sensor (WS)-based [11]. More specifically, MV-based uses gait's visual pattern, FS-based analyzes dynamic weight/pressure distribution, and WS-based uses acceleration patterns for identification [14]. Unlike MV and FS-based approaches where the system will likely require additional hardware such as cameras or integrated floor's sensors, with the development of the current society, where almost everybody will carry at least one wearable smart device such as a fitness-tracker or smart-watch, WS-based in most cases, will not require any additional hardware installation. Also, WS-based authentication is a very helpful way to improve authentication in electronic devices since it does not require explicit user interaction and the authentication can be done continuously and passively. A compelling example is smartphone-based authentication using WS-based gait identification. WS-based approach can use data from different sensor types recorded from the device such as accelelerometer (measures the acceleration), gyroscope (measures the rotation), and force sensor (measures the force when walking) [14]. However, most of the recent research literatures have put their focus on accelerometer-based gait identification [14, 15, 16, 8, 17, 1]. Therefore,

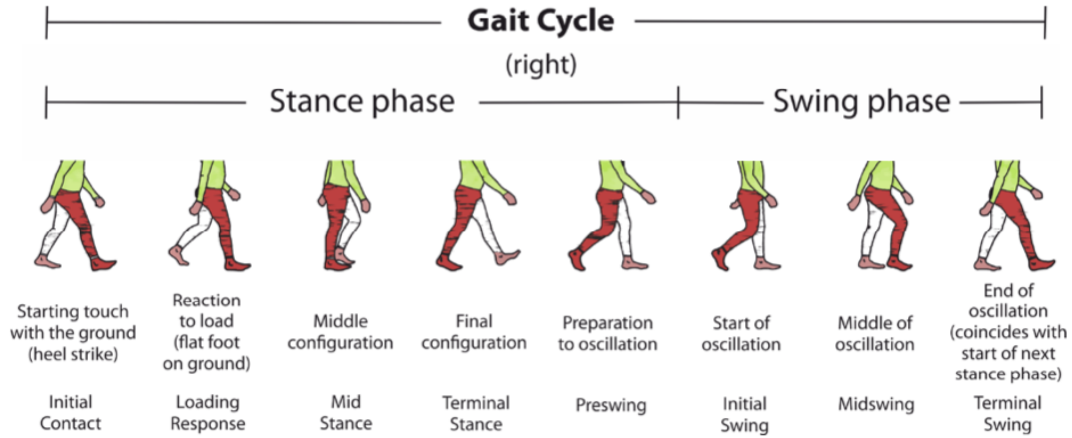


Figure 1: Gait cycle dynamics [13].

accelerometer-based approach is considered one of the most dominant direction in gait classification research and development.

2.2 Smartphone-based gait authentication using deep learning

Smartphone-based gait authentication is done by collecting and analyzing motion sensor data from smartphone's hardware sensors. Figure 2 demonstrates the coordinate system of 3-dimension motion sensor data (accelerometer sensor, gyroscope sensor) of an Android device and Figure 3 shows an example of a recorded 8 steps of accelerometer data from an Android smartphone [18]. After sensors data being collected, performing gait analysis and gait classification using different ML techniques will be the next step for solving the gait biometric authentication problem.

With the current breakthroughs that Deep Learning (DL) has brought in in other application domains such as speech recognition [20] and computer vision [21], it is understandable that trying to apply deep learning techniques in gait classification is a natural next step for enhancing gait biometric authentication.

It is worth noting that, solving gait recognition problem itself is not a new problem as several approaches have been proposed, such as either using signal matching algorithms like Dynamic Time Warping (DTW) and its variations or using different classical ML techniques such as k Nearest Neighbors (KNN), Support Vector Machine (SVM), or Hidden Markov Model (HMM) [11]. However, more recently, the approach of using DL model and solving gait recognition as a classification problem has emerged as one of the main research focuses in gait biometric [1, 11, 22]. More importantly, some of the reported results are quite remarkable. For example, compared to the results of using manual extracted features in which the accuracy of the ML classifier is 71% [1], experiment results of one of the proposed methods

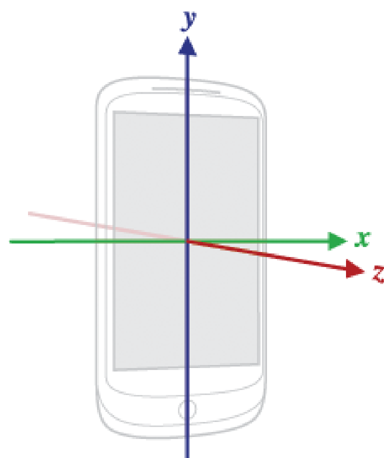


Figure 2: Standard 3-axis coordinate system defined relative to the device's screen (Android). [19]

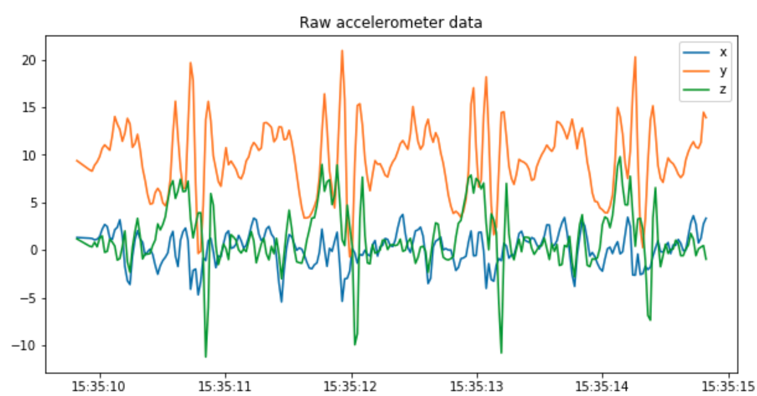


Figure 3: Raw-accelerometer data from a short walking session (sample data is taken from IDNet gait dataset [18]).

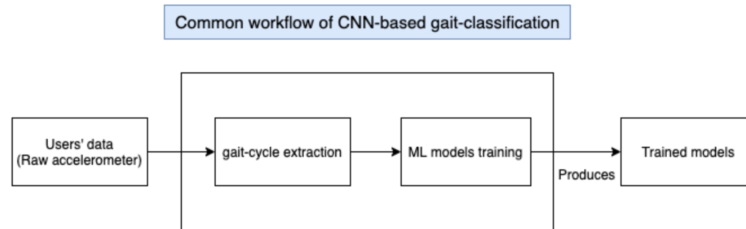


Figure 4: Common workflow of DL-based gait classification approaches.

achieves over 93.5% and 93.7% accuracy in person identification and authentication, respectively [22].

Throughout the proposed DL solutions, the common steps for solving the problem are gait segmentation and gait classification. Gait segmentation can also be called as gait cycle extraction and gait classification is the actual step in which ML modelling is done for classifying the extracted gait cycles. Figure 4 shows a common simplified version of signal processing workflow for solving gait classification problems across the papers [1, 11, 23, 24].

Although Figure 4 illustrates a high-level description of the data processing steps, detailed solutions from each paper are not identical. For example, Matteo et al. include additional data pre-processing steps such as *filtering*, and *orientation independent transformation* before gait cycle segmentation or *normalization* after gait cycle segmentation, before feeding the extracted gait cycles into the actual ML classifier [1]. Or Qin Zou et al. propose another additional step called *gait data extraction* before the actual gait cycle extraction step to filter out non-walking data. It is worth noting that the proposed solution from Matteo Gadaleta uses Convolutional Neural Network (CNN) model as an automatic featurizer before using the shallow classifier such as SVM for solving classification problem, while Qin Zou utilizes DL techniques in a different way, in which both CNN and Recurrent Neural Network (RNN) are combined to make authentication decision [22].

Since gait segmentation and gait classification (gait-recognition) appear to be the common steps across the proposed DL-based solutions [1, 22, 11, 25], it is worth reviewing the current state-of-the-art techniques and examples in these two common steps.

2.2.1 Gait segmentation

Gait segmentation is a crucial step in gait classification as the output of this step is the input of the classification step. Also, since ML model's decisions are highly dependent on the quality of training data (garbage in, garbage out) [26], it is important to get this step done correctly and with a high accuracy. The current state-of-the-art approaches can be categorized into 3 different approaches: *Cycle Segmentation Based on Cycle Extremes Identification*, *Cycle Segmentation Based on Cycle Length Estimation*, and *Step Segmentation Based on the Estimated Number of Steps* [11].

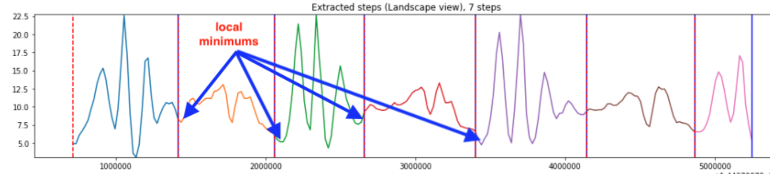
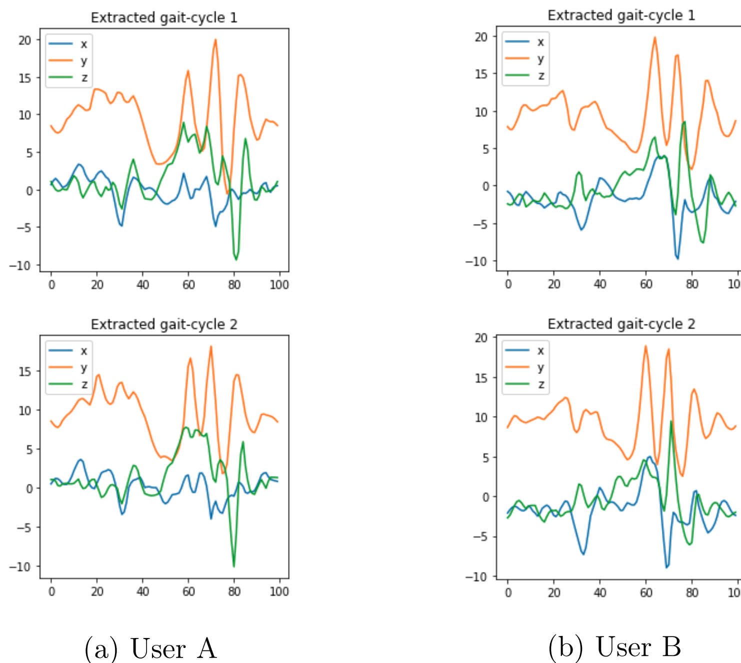


Figure 5: Step starts and ends with local minimum.

Specifically, Cycle Segmentation Based on Cycle Extremes Identification [27] segments gait cycle based on the observation that each step usually starts and ends with local minimum and one gait cycle consists of 2 consecutive steps (as shown in Figure 5). Cycle Segmentation Based on Cycle Length Estimation [28] utilizes the fact from the former approach as well as the cycle length estimation. Unlike the other two approaches, Step Segmentation Based on the Estimated Number of Steps [29] helps to reduce the complexity and computational expensiveness as well as producing a higher performance by looking at the signal on y-axis only and doing segmentation by doing thresholding combined with *prior knowledge* of the number of steps from the signal. However, the requirement of prior knowledge of the number of steps appeared to be an obstacle for using this approach in the real world, especially when the number of steps is not easily accessible [30, 11].



(a) User A

(b) User B

Figure 6: Extracted gait cycles (accelerometer) from two random users A and B.

As an example, Figure 6 shows the visualization of the extracted gait cycles from 2 random users in IDNet gait dataset [18]. Specifically, x, y, and z axes are the measurement of the phone's acceleration while the user was walking. It is worth noting that, x, y, and z values are acceleration including gravity and collected from the hardware sensor.

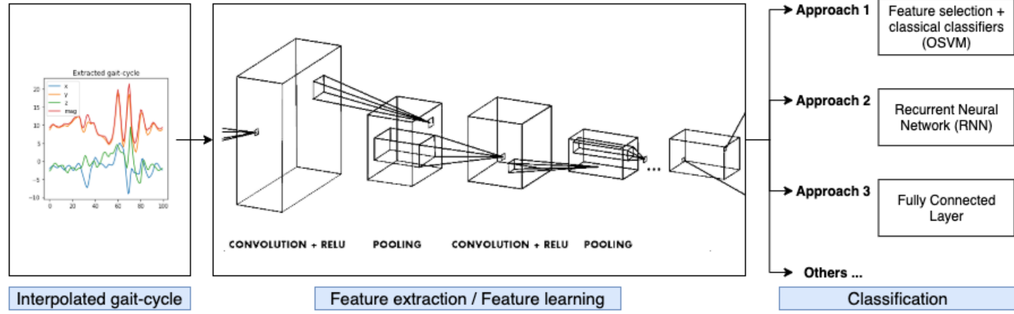


Figure 7: Summary of approaches using CNN-based gait classification.

2.2.2 Gait classification

Gait classification, put in other words, is a step to translate authentication problem into a classic ML problem: classification problem. Unlike the previous works, in which the solutions can either be computationally expensive, require long authentication phase (Hidden Markow Model requires up to 30 seconds of data to achieve a good performance) [1], or require advanced feature engineering techniques [25, 1], the main contributions from the state-of-the-art DL-based approaches are that DL-based models provide a good accuracy and there is no need to require handcrafted features engineering [1, 31, 2].

More specifically, based on the fact that CNNs are very good at features embedding and require a minimal effort from human intelligence to extract meaningful features in image processing [32], DL-based solutions in gait classification use CNNs to achieve the same ultimate goal [1, 31, 2]. Although the detailed model architectures are different in the current state-of-the-art papers, the main idea remains consistent: to use a set of different layers such as Convolutional, ReLU, or Max pooling layers to extract motion characteristics.

Due to the nature of CNN, which requires a fixed shape of input data (for example, in image processing an input shape of an RGB 256x256 pixel image should be *(256, 256, 3)*), it is necessary to convert from the dynamic shapes of actual gait cycles (depending on how fast/slow a person walks) to a fixed length shape. This can be done by using interpolation. As a result, the input data for CNN network are *interpolated gait cycles*, not the actual gait cycles extracted from the segmentation step [1, 31, 2].

After feature extraction is done by using CNN, several different approaches have been applied to solve the classification problem. Figure 7 gives a summary of different ways of doing classification. Matteo et al. [1] propose a solution for taking the output from CNN, applying dimensionality reduction using Principle Component Analysis (PCA), and passing the reduced vector to a One-class Support Vector Machine (OSVM) classifier. More recently, Shuochao Yao et al [2] introduce *DeepSense*, a Unified Deep Learning Framework in which user's identification problem is considered as both regression and classification ML problem. To solve this issue, the

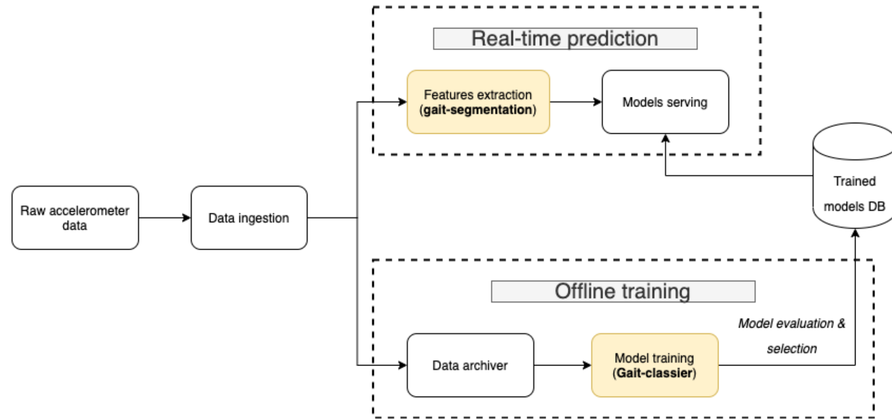


Figure 8: High-level ML pipeline for CNN-based gait authentication.

proposed framework uses output from convolutional layers as input for other recurrent layers in the model architecture.

2.3 Gait authentication productionization

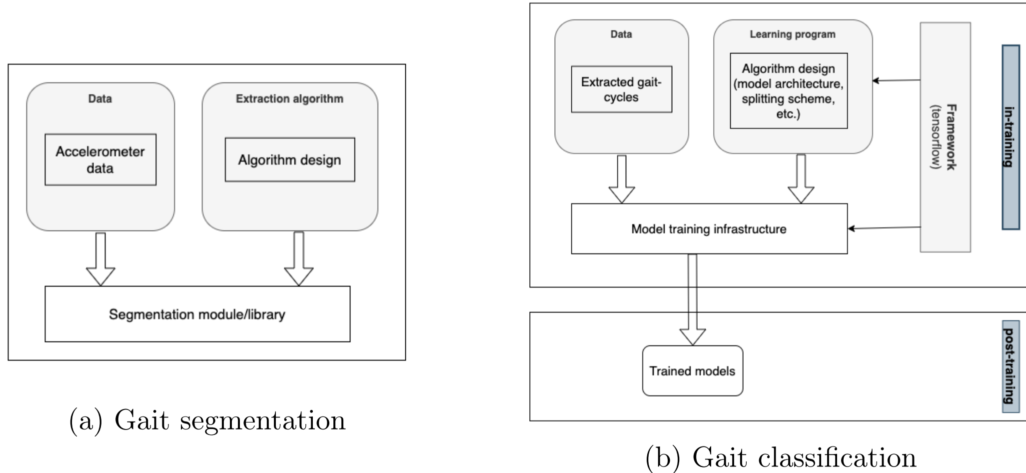
For simplicity and because of the fact that the main goal of this thesis is about ML testing, not the actual ML modelling, this section will make an assumption that gait authentication's data processing pipeline consists of two main parts: gait segmentation and gait classification (as shown in Figure 4).

Figure 8 presents a high-level architecture design for what the production pipeline should be in order to support the data processing workflow described in Figure 4. Basically, after raw acceleromter data is ingested into the system, the data will be streamed into two different services: *real-time prediction* and *offline-training*. Real-time prediction service is responsible for producing real-time predictions from the input signal, and the other component is responsible for archiving historical data and doing offline training. There are two highlighted components that are worth paying attention to, which are *Feature extraction* and *Model training*. These components are the most distinctive and domain-dependent components compared to other standard ML pipelines (data ingestion, models serving, and data archiver are the common but non-negligible components that are needed for building an ML software system [33]).

Figure 9 provides microscopic views of the two main ML-heavy system components in Figure 8, by demonstrating the components involved in each of them. As can be seen from Figure 9a, gait segmentation is a self-contained software component, which takes data input as accelerometer data, and outputs a set of extracted gait cycles. Unlike gait segmentation, gait classification contains several other non-traditional software components such as model training procedures (in-training) and a set of trained models (in post-training) as shown in Figure 9b.

It is worth noting that, within the scope of this thesis, performing ML testing for

gait authentication is to perform necessary testing on these two components. Also, doing ML testing is more than just performing models evaluation on the trained ML models.



(a) Gait segmentation

(b) Gait classification

Figure 9: ML components involved in CNN-based gait authentication.

3 Traditional system testing vs ML-based system testing

3.1 Traditional software testing and its limitations when applying to an ML system

Software testing is a broad term encompassing a wide range of different activities, from the testing of a small piece of code by the developer (unit testing), to the customer validation of a large information system (acceptance testing), or to the monitoring at run-time of a network-centric service-oriented application. Software testing has always been a crucial part of the software development process, as it is used to ensure the quality, the specification compliance and to identify possible malfunctions of the system. Besides being widely adopted by the industry for quality assurance, software testing has also attracted a lot of interests from researchers in recent years [34].

Together with the rapid growth of ML applications, there is a need to develop a proper testing methodology to ensure the trustworthiness of the system. For example, some safety-critical applications such as self-driving car, and medical treatments require a critical testing in terms of correctness and reliability, since a mistake in these application domains can cause serious consequences [6]. Ideally, there is a strong desire to apply software testing into an ML system to expose problems and potentially facilitate to improve the reliability of the ML systems.

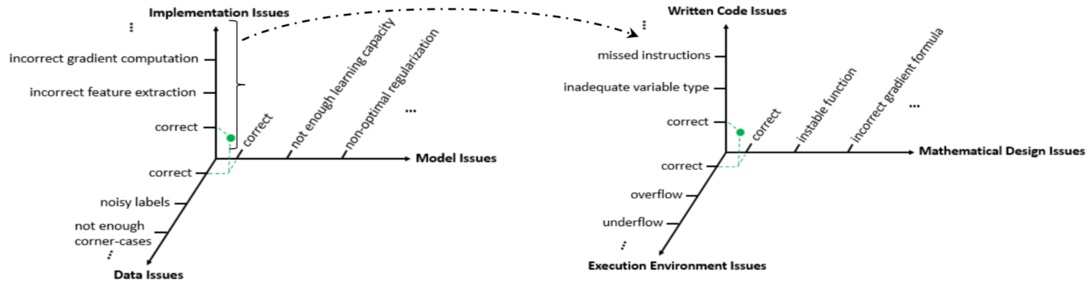


Figure 10: Test space in ML systems [5].

However, due to some fundamental discrepancy between traditional software and ML software, applying traditional software testing is shown to be not a sufficient way to accomplish that [6, 5]. Specifically, a normal software system is a translation of user stories and all required system specifications that are well-defined, and therefore, the test oracles are consistent. Line coverage, branch coverage, data flow coverage are shown to be sufficient ways to measure test adequacy [6]. Nevertheless, unlike traditional software, ML system is non-deterministic and the decisions are highly dependent on the data and the algorithms that the models were trained on. More specifically, due to the statistical nature of ML models, expected test outputs are not always easy to determine, and having a small portion of incorrect predictions out of hundreds of test samples does not invalidate the validity of a statistical model. Also, existing test adequacy metrics such as line coverage do not guarantee the correctness of statistical models [35, 6, 5].

As Murphy et al. [36] state that, ML is difficult to test because it is used to answer the questions that have not been answered before. Moreover, ML bug might derive from many different factors such as data issues, model issues or implementation issues. Figure 10 presents a comprehensive view of the test space in an ML system for identifying possible faults in ML programs. Compared with traditional software, the dimension and potential testing space of an ML program is much larger [5]. Specifically, apart from the common spaces where bugs might occur in both software and ML system such as implementation, written code, and execution environment, in an ML system, bugs can also occur in other dimensions such as data, model, and mathematical design.

Due to the non-deterministic nature of ML models and the complexity of the testing space, current existing software development techniques must be revisited and adapted to provide proper testing approaches for ML systems [5].

3.2 Current ML testing techniques

While developing an ML application based on requirements, traditional software testing can be applied. However, not all ML applications can give a clear definition

on how the system should behave in all application scenarios. For example, in autonomous driving, there are an infinite number of cases that the model has to make a decision, and therefore, new software testing approaches need to be developed. Overall, the current proposing solutions mainly focus on testing three areas: *detecting bug in data*, *detecting bug in ML models* (learning program), and *detecting bug in a framework* [6, 5].

3.2.1 Bug detection in data

Bug detection in data checks several criteria such as whether the data is sufficient for train and test, whether the data is representative of future data, whether the data has noise, or whether the data was poisoned [6]. The common approach for detecting bug in data is to aggregate some statistical features (min, max, mean, and median) from the data and use that as a validation check. Recently, *Tensorflow Extended* (TFX) introduced a feature called skew detector to check the similarity between train and test data [37]. Also, Hynes et al. introduced *data-linter*, a *code-linters like* library to perform sanity check on the data. Basically, data-linter covers 3 main problems: miscoded data (typing mismatch), outliers and scaling (uncommon list length), and packaging errors (duplicate values, empty example) [38, 6].

3.2.2 Bug detection in ML models

For bug detection in ML models, there are two separate aspects that errors can occur: *conceptual errors* and *implementation errors*.

Approaches for testing conceptual errors make an assumption that the implementation was done correctly, and the testing focuses on detecting potential issues in the calibration of the models. The approaches for testing conceptual errors can be divided into two groups: *black-box testing* and *white-box testing*. Black-box testing is a testing technique that treats ML models as a black-box and it does not need to know the internal architecture of the model. The main goal of this is to make sure the model provides a high accuracy, without worrying about its internal parameters. The common denomitator of black-box testing approaches is the generation of adverserial test data. Specifically, test data can be drawn from some generative models or by adding some pertubation or making some modification such as changing some pixels in images, applying spatial transformations [5]. One major limitation of black-box testing is the generation of adversarial data. For example, data generated from generative models assume that test data and train data are from the same distribution, which makes the ML system vulnerable with respect to malicious adaptive adversaries [5]. Also, since synthetic datasets are usually created by adding some tiny, undetectable pertubations (since any major change would require manual inspection), the representativeness of adversarial data compared to real world setting is also an issue. As a result, synthetic test data cannot cover all the behaviors of the model, even after performing a large number of tests. To help overcome these issues, white-box testing was introduced as a technique which takes

a model's internal structure as a part of driving decisions to generate more relevant test cases. With the current development of Deep Neural Networks (DNNs), recent research are mainly focusing on white-box testing DNNs. Pei et al. introduce DeepXplore as the first white-box testing framework for DNNs [4]. The main contribution of DeepXplore is that it can automatically identify erroneous behaviors of DL models without the need of manual labelling. In addition to this, DeepXplore introduce a new testing metric called *neural coverage*, a *code-coverage* influenced, that is used to measure the amount of neurons triggered by a set of input data. In practice, DeepXplore applies differential testing, a pseudo-oracle testing approach in traditional software testing [39], with the idea of finding a large number of difference-inducing inputs while maximizing neuron-coverage. This testing approach can be formulated as a joint optimization problem [4, 5]. Domain-specific constraints are added to generate test data (changing background color, add rain, etc.), and the generated test data are kept to co-train with the actual train data to improve model robustness [4]. More recently, DeepGauge, DeepTest, DeepRoad are also introduced as enhancements for DeepXplore with neuron coverage still used as a test adequacy metric [5].

Because many ML algorithms use randomness to solve optimization problems, their nature is stochastic [40]. As a consequence, most existing software testing techniques are not fully compatible with an ML system. Recently, the ML community have introduced a handful of different testing techniques such as *numerical-based testing*, *property-based testing*, *metamorphic testing*, *coverage-guided fuzzing testing*, and *proof-based testing* to detect implementation errors [5]. In numerical-based testing, developers usually check the accuracy of gradients using a finite difference technique. Property-based testing is a technique that consists of inferring the properties of a computation using the theory and formulating invariants that should be satisfied by the code. Metamorphic testing defines a set of *Metamorphic Relationships* which then can be used to test the applications. Mutation testing involves the process of injecting mutant (artificial faults) in a program under test and generating test cases to detect them. For example, DeepMutation developed by Ma et al. adopts the mutation testing technique to detect ML bugs in Deep Learning systems [41]. *TensorFuzz* developed by Augustus Odena and Ian Goodfellow is an example of a testing framework using the coverage-guided fuzzing technique [42]. The fuzzing process consists of handling an input corpus that evolves through the execution of tests by applying random mutation operations on its contained data and keeping only interesting instances that allow triggering new program behaviors [42, 5]. Lastly, proof-based testing is a technique that requires a formal specification and formal proofs of written theorems which define what it means for the system to be correct and error-free. This technique helps verifying the mathematical correctness without human-intervention [43].

3.2.3 Bug detection in the framework

Bug detection in the framework focuses on testing third-party ML frameworks and the algorithm implementation using the frameworks.

For framework testing, security vulnerabilities, runtime behavior, training accuracy, and robustness are the main aspects that attract research interests. Specifically, Xiao et al. [44] point out some security risks by studying framework testing on common DL frameworks such as Caffe, Tensorflow, and Torch. Some issues that were raised and confirmed by developers include memory-overflow, and use-after-free. As a consequence, this might lead to possible attacks such as denial-of-service or control-flow hijacking attacks. Besides, Guo et al. [45] performed training accuracy and robustness testing on different framework including Tensorflow, Theano, and Torch, and the results show that runtime training behaviors are different in each framework, but the accuracy remains similar.

Incorrect algorithm implementation may not cause crashes, errors, or efficiency problems [6]. For instance, Cheng et al. injected implementation bugs into ML code in Weka and it was found out that from 8% to 40% of the injected bugs were statistically indistinguishable from the original version [46]. As a result, it is challenging to detect these subtle issues. Differential testing using different implementations of the same algorithm or metamorphic relations are the common techniques to detect bugs in the ML code implementation [5, 6].

4 Proposed testing solution

Although ML testing has emerged to be a necessary task to produce robust and reliable AI systems, most of the research focuses are about testing image related datasets and its related use cases. For example, the proposed testing frameworks such as DeepXplore uses image datasets like MNIST or ImageNet [4]. Other application domains such as gait classification have not attracted much interest from the community. Also, due to the significant difference between gait data and image data (time series data vs. image data), a direct application of the proposed testing solutions is also insufficient (e.g. some domain-specific constraints such as changing weather condition or background image in metamorphic testing in autonomous driving are not directly transferable). Therefore, this section is going to propose a solution for performing tests on a gait classification production system, from the organization of related workflow in ML testing to detailed testing proposals for gait segmentation and gait classification.

4.1 Organization of related work and workflow in ML testing

First of all, ML testing is not a completely new work, but is rather considered as an extension of traditional software testing to assure the quality of ML applications.

Therefore, applying traditional software testing with modifications is a way to organize the related work. This thesis uses a combination of V-model in traditional software testing and the three questions: *where to test*, *what to test*, and *how to test* as the backbone to organize the workflow in ML testing in general, and in testing gait authentication system as a specific use case. Overall, V-model is used as a high-level view to identify the testing phases that need to execute. For each phase, the answers for the three questions are used to identify the specific tasks that need to be done.

4.1.1 V-model in software testing

As can be seen from Figure 11, V-model consists of two phases: *Verification* and *Validation*. It is based on the association of a testing phase for each corresponding development phase. Verification involving static analysis technique (review) is done without executing code. It is the process of evaluation of the product development phase to find whether specified requirements meet. Validation involving dynamic analysis technique (functional, non-functional), performs testing by executing code. Validation is the process to evaluate the software after the completion of the development phase to determine whether the software meets the customer expectations and requirements [7]. In V-model, testing activities are within the Validation Phase. There are four main testing phases: *Unit Testing*, *Component Testing*, *System Integration Testing*, and *Acceptance Testing*. Unit testing is used to verify that smallest entities such as program module can function correctly when isolated from other components. In the V-model, unit test plans (UTP) are developed during module design phase and the test plans are being executed to detect errors at the code or unit level. Unlike unit testing which tests individual programs or modules of a program, component testing is a type of software testing in which usability and behavioral evaluation of each individual software component is tested. While unit testing needs to know the internal architecture of the software, component testing treats each software component as a black-box and it is performed once the unit testing is performed. Unit testing and component testing are the phases that are used to test each component independently. However, system testing and acceptance testing are designed to test the system as a whole. System testing tests the functional and non-functional requirements of the developed application. Acceptance testing verifies that the delivered system meets user's requirements and the system is ready for use in real world.

4.1.2 Where, What, How to test?

As shown in Figure 10, the testing space in ML systems is relatively complicated and ML bugs can occur in many different phases. Therefore, having a set of questions while performing testing can help reducing the complexity and organizing the related work and testing activities.

Firstly, *"Where to test?"* is a question that can help to identify the pieces that

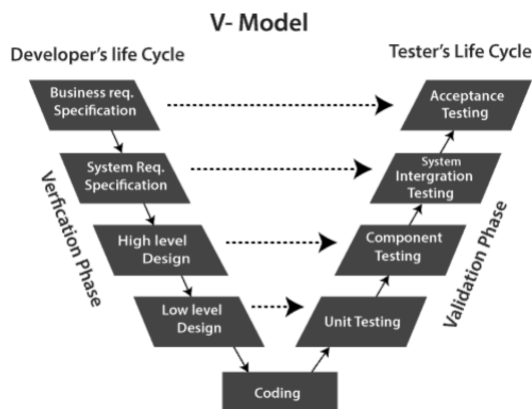


Figure 11: V-model in Software Testing [7].

need to be tested. Put in our ML testing context where an ML related software component needs to be tested, the required testing pieces (testing components) can be listed as follow: data testing, learning program testing, and framework testing.

After identifying the pieces that need to be tested, answering the next question: *"What to test?"* can help to identify a set of properties that are cared for testing. For ML testing, a set of common testing properties are: *correctness*, *robustness*, *efficiency*, *fairness*, *interpretability*, *security*, and *privacy* [6].

Finally, answering the question *"How to test?"* can help to organize the detailed workflow of the testing activities for each component and each property that needs to be tested. Figure 12 shows an ideal workflow for performing ML testing in an ML system. Basically, testing activities are divided into two groups: *offline testing* and *online testing*. Offline testing involves the testing activities that are done before the model deployment process to conduct requirement analysis to define the expectations of the users for the ML system under test. Test generation, test evaluation, bug report analysis, debug and repair are parts of offline testing. If all bugs are repaired and the regression test does not expose other new bugs, the offline testing process ends, and the model is deployed. Although offline testing is a good way to identify and correct problems before deploying models to customers, there are several reasons that make online-testing essential. First, offline-testing usually relies on test data, which is historical data and does not fully represent future data. Similarly, offline-testing cannot replicate some problematic issues that can happen in real applied scenarios, such as data loss or system delays. Besides, online-testing can help monitoring real-life performance and can help detecting any abnormal performance drop so that the system can take subsequent steps such as doing model retraining or further advanced debugging [6, 47].

Although online-testing is undoubtedly a useful way for testing system performance in a real-world setting, it is non-trivial and exhaustive to set up the experiments and monitor the results, and therefore, this thesis will only focus on offline testing activities in a gait authentication system. Online testing can be considered as future

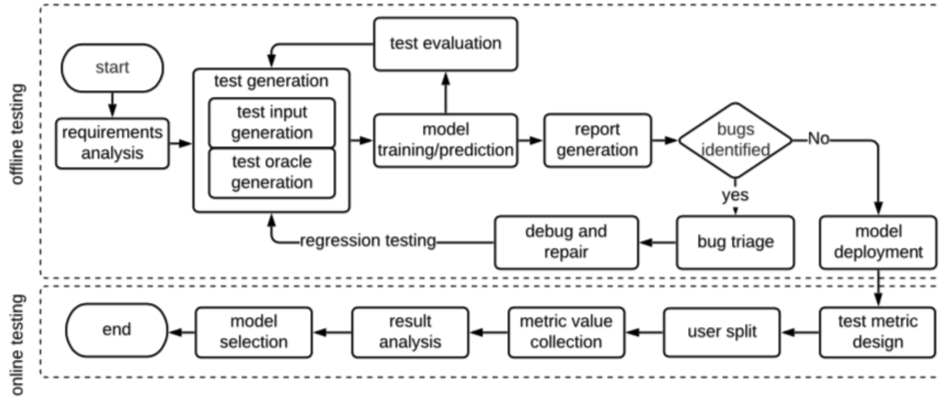


Figure 12: An ideal ML testing workflow of an ML system [6].

work.

4.2 Gait segmentation testing

As shown in Figure 9a, gait segmentation consists of two main components: Data component and Extraction algorithm (learning program) component. Compared to the *"Where to test?"* question presented in the previous sub-section, Data testing and Learning program testing are the essential pieces that need to perform in gait segmentation testing. Since gait segmentation is self-contained and does not require ML models training, third-party ML framework testing is not necessary and can be neglected.

4.2.1 Data testing

Data input for gait segmentation is raw 3D accelerometer sensor data. However, some potential bugs might occur in smartphone-based accelerometer data such as broken accelerometer data (e.g. invalid values or inconsistent sampling rates) [48]. Therefore, necessary testing needs to be done to filter out these issues. Some common practice that can be applied to expose such issues are to define a list of domain-specific constraints (e.g. upper and lower bound of accelerometer values), or filter out invalid values (e.g. NaN values). In addition, although both Android and iOS support sensor data collection at a specific sampling rate [49, 50], since accelerometer data are data collected from hardware sensor, there is a possibility that data are not sampled correctly. Therefore, re-validating sampling rates by checking absolute timestamps between datapoints from accelerometer data is also an important step.

4.2.2 Extraction program testing

There are two phases that the testing activities should focus on in this component: *unit testing* and *component testing.* Depending on the actual implementation and algorithm, proper unit test plans should be designed and executed in unit testing. Code coverage can be used as the evaluation metric. For component testing, several properties should be focused, especially *correctness* and *efficiency.* Since extraction program is a step to process data as input to the classifier, it does not have a considerable impact on other properties such as interpretability, fairness, and privacy. As a result, these properties can be neglected.

Correctness. By definition, correctness measures the probability that the system under test "gets things right" [6]. Put in our context, the following metrics can be developed to measure the correctness of the extractor: *recall*, *precision*, and *alignment accuracy.*

For gait segmentation, recall is defined as follow:

$$Recall = \frac{TP}{TP + FN} \tag{1}$$

in which True Positive (TP) is the number of gait cycles that the program is able to extract with reference to the ground truth gait cycles and False Negative (FN) is the number of gait cycles that are not extracted by the program even though they appear in ground truth labels.

The same convention can be applied for precision, with the formula defined as follow:

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

in which False Positive (FP) is the number of gait cycles that the program extracts but are in fact not real gait cycles.

While recall and precision are indicators to tell the fraction of the total amount of relevant instances that were actually retrieved, they do not tell how precise or accurate the extracted gait cycles are compared to the ground-truth gait cycles. Alignment accuracy is a metric to help indicating that property. Specifically, alignment accuracy compares the timestamps between the extract gait cycles and the ground truth gait cycles and calculate the overlapping zone as the alignment accuracy. Since there has not been a proper formula that can address this alignment measurement, this thesis proposes a formula for measuring it. Specifically, the alignment rate is calculated as follow:

$$Alignment_rate = 1 - \frac{\left|gc_{1_{start_time}} - gc_{2_{start_time}}\right| + \left|gc_{1_{end_time}} - gc_{2_{end_time}}\right|}{gc_{1_{duration}} + gc_{2_{duration}}} \tag{3}$$

in which *gc_1* and *gc_2* are candidate gait cycle and ground truth gait cycle respectively.

Intuitively, if an extracted gait cycle has the exact timestamps with the ground truth one, the alignment accuracy will be 100%. If they are far apart, the alignment accuracy will be closer to 0.

Even though computing the above mentioned metrics is a useful way to test the correctness of the algorithm, there is an issue with that approach, which is the expensiveness of the manual process for labelling ground truth gait cycles. Therefore, in addition to the above metrics, this thesis suggests an additional step which uses differential testing, a semi-oracle testing technique from traditional software testing to identify potential issues and correctness in the extractor. More specifically, based on an assumption that given the same input data, different extractors should extract the same gait cycles. If there is a discrepancy in the outputs, it means one of the extractors has faulty error, or there are some unknown issues with the test input data. By logging the test outputs from these test cases, developers can debug and do further investigation, which can help identifying potential unseen issues and corner cases from the extractor. One might argue that differential testing is not an ideal approach since it could have been that both candidate extractors can have the same bug, and therefore, some successful test cases might still contain some hidden errors. It is admitted that this is a correct argument, however, compared to ground truth testing, differential testing does not require manual labelling and this testing technique does not need to cover absolutely every corner cases. It is rather considered as a technique to uncover issues from the failed cases that it discovers.

Efficiency. As mentioned in section 2.2.1, gait segmentation algorithms can use different techniques and solutions, some of which might be more expensive to run than others. Therefore, having some quantifiable measurements is a good way to evaluate the extraction runtime performance. This thesis introduces *extraction runtime*, a scalar value to present the efficiency of the extractor. Basically, extraction runtime is the average time that takes to process one second of data, and the unit is *ms/second of data*. Since the number of extracted gait cycles depends on the extractor, the fairest test is to measure against the ground truth gait cycles on a labelled test dataset.

4.3 Gait classification testing

As summarized in section 3.2.3, various testing techniques are being proposed and used for testing common DL frameworks such as Tensorflow and Pytorch. Furthermore, all of these frameworks support DL algorithms design and implementation, including time series data classification [51, 52]. Therefore, this section will not cover third-party DL framework testing, but it will rather focus on other aspects.

Based on the detailed view of components involved in gait classification shown in Figure 9b, *Data*, *Learning program*, *Model training infrastructure*, and *Trained models* are the pieces that need to be tested.

Table 1: Data quality constraints for extracted gait cycles.

Dimension	Constraint	Argument(s)	Semantic
Completeness	*is_complete*	gait cycle	check that there are no missing values in the gait cycle (e.g. NaN values)
Consistency	*has_input_shape*	gait cycle	has the expected input shape
	is_in_range	gait cycle	validation of the fraction of values that are in a valid range
	has_valid_duration	gait cycle	if the time interval of a gait cycle is longer than x seconds, it might give an indication that something might have gone wrong in the extraction step

4.3.1 Data testing

Unlike gait segmentation where data input is the raw accelerometer data collected from mobile devices, data input for gait classification is the interpolated extracted gait cycles from the gait segmentation job. As a result, data quality is highly dependent on the data provided from the segmentation step. Furthermore, trained models rely heavily on the input data and subtle errors caused by the train data can be really hard to detect [5]. Therefore, testing activities should be more comprehensive and thoughtful in this step compared to data testing in gait segmentation. For data testing, Unit Testing and Component Testing are the main validation phases.

Unit testing. “Unit tests” for data testing consist of a set of constraints (rule-based detection) that the input data has to comply with. The list of constraints can evolve over time. The focus is about the definition of the checks and validations, not the computation of the metrics required for constraints [53]. Data quality constrains are developed based on different data quality dimensions such as *completeness* and *consistency*. Completeness refers to the degree to which an entity includes data required to describe a real-world object. Consistency is defined as the degree to which a set of semantic rules are violated. Table 1 presents a set of constraints that can be used to validate data input.

Component testing. Unlike Unit testing, component testing focuses on computing

Table 2: Testing metrics for different data quality dimensions.

Dimension	Metric	Semantic
Completeness	*completeness*	the fraction of non-missing values gait cycles
Consistency	*compliance*	ratio of gait cycles matching predicate

Table 3: Test cases for testing extracted gait cycles.

Test case	Argument(s)	Assertion
gcs_are_complete	set of gait cycles (training data / test data per user)	*completeness(gait cycle) == 1.0*
gcs_have_input_shape	set of gait cycles (training data / test data per user)	*compliance(has_input_shape(gait cycle)) == 1.0*
gcs_are_in_range	set of gait cycles (training data / test data per user)	*compliance(is_in_range(gait cycle)) == 1.0*
gcs_have_valid_duration	set of gait cycles (training data / test data per user)	*compliance(has_valid_duration(gait cycle)) == 1.0*

Note: *gcs* is a shorthand for *gait cycles.*

testing metrics and performing testing on datasets as a whole. Therefore, *Translating constraints to metrics computations* and *Test oracle generation and actual execution* are parts of component testing.

While performing testing, a set of metrics related to different data quality dimensions should also be considered in computing. Table 2 shows the metrics for measuring different data quality dimensions.

Based on the suggested constraints and metrics, the list of test oracles can be constructed in Table 3.

Last but not least, skew detection between train and test data can also be considered as a “component test” in data testing. Specifically, the idea is that the training instances and the instances that the model predicts should exhibit consistent features and distribution. Skew detection can be done by analyzing the distribution of statistical features from train and test data.

4.3.2 Learning program testing

Performing unit test in learning program is sufficient as the main goal of performing the test in this component is to test the correctness of our implementation (model architecture, callbacks, etc.).

Test adequacy metric for this can be code coverage.

4.3.3 Model training infrastructure testing

This is the most non-ML related component and therefore, applying traditional software testing workflow should be sufficient (mainly unit testing).

Test adequacy metric for this is code coverage.

4.3.4 Trained models testing

Besides data testing, this is inarguably the most time-consuming component to perform tests. ML properties such as correctness, efficiency are connected with the behavior of the trained ML models. Therefore, these properties should be the focus of testing in this component.

Unit testing. For unit testing, despite some suggestions and ideas for doing white-box testing in DL models (DeepXplore, TensorFuzz), there is not yet a mature way to perform white-box tests in our particular use case (DL model using human-gait data). Therefore, within the scope of this thesis, this section can be seen as a future work. Within this thesis, the model will be treated as an isolated component, and the testing is done on component level.

Component testing. For component test, the list of suggested ML properties should be tested are: *correctness*, *robustness*, *efficiency*, *fairness*, *interpretability*, *security*, and *privacy*. Although other metrics such as fairness, interpretability, and privacy are useful to have, correctness, robustness, and efficiency can help to bring more immediate visibility to the product's performance since they can give a good estimate of how well the system can work and how efficient the component can run in the software system. As a result, these metrics are the main focuses that will be addressed in this thesis. Other metrics can be considered as future work

- **Correctness**. Correctness measures the probability that the ML system under test 'gets things right'. In biometric authentication in general, correctness can be measured by the following metrics: TAR, and FRR [54]. Therefore, to align with biometric authentication's jargon, these metrics can also be used to measure the correctness of gait authentication. Technically, TAR is equivalent to the True Positive Rate (TPR) and FRR is equivalent to True Negative Rate (TNR). Although these rates can tell how well a model performs in both positive and negative test samples, one thing to note is that, it can only tell model correctness at a hard-coded decision threshold. For example, the returned probability from the classifier is a number between 0 and 1, but there should be a certain threshold that the system can say if the value is higher than the threshold, it is predicted as the expected user, otherwise, that is not the user. As a result, by only looking at TAR and FRR, one piece of missing information here is how well the probabilities from the positive classes

are separated from the negative classes without explicit thresholding. This information can be achieved by computing *Receiver operating characteristic - Area Under Curve score* (ROC - AUC score). Therefore, there are three main metrics that contribute to correctness property: TAR, FRR, and ROC - AUC score.

- **Robustness**. Robustness measures the resilience of an ML system's correctness in the presence of perturbations. Robustness can be measured by conducting adversarial perturbation on input data. For the sake of completeness, both random data and constant data with proper input shape will also be used for testing.

- **Efficiency**. The efficiency of an ML system refers to its construction or prediction speed. Put it in perspective, the two metrics that catch our attention are: training time and inference time. For inference time, the performance should be measured on the actual environment where the production model performs (for example, if model serving uses CPU, the actual testing has to be done on the same hardware setup as well). The unit for inference runtime is *ms per prediction.*

5 Experimental materials and procedure

5.1 IDNet gait dataset

IDNet gait dataset is a dataset that contains motion data from 50 subjects collected from Android devices worn in right front pocket, and the collection was done during a period of six months. It is publicly available at the Department of Information Engineering, University of Padova's website. For each subject, several five-minute sessions were recorded in different conditions such as different shoes and clothes with a natural walking mode. Data collection was done by using an Android inertial data logger application. And different sensors data were collected at the same time. The list of collected sensors are:

- Gyroscope
- Magnetometer
- Accelerometer
- Linear accelerometer
- Rotation vector

Unlike gyroscope, magnetometer, and linear accelerometer which were collected directly from hardware sensors, linear accelerometer and roation vector are software sensors that were evaluated and provided by Android API. Linear accelerometer

is the main sensor data used for gait segmentation testing. For gait classification testing, gyroscope will be used together with linear accelerometer data since the DL model used for testing requires data from both sensors.

Also, it is worth noting that, the sampling rate on IDNet gait dataset is non-uniform and is different for each sensor. Besides, data pre-processing had already been done before the dataset was generated [1].

5.2 Experimental procedure

The experiments are grouped into two main sections: gait segmentation testing and gait classification testing. Since IDNet gait dataset has already performed data pre-processing, data testing requires another set of data. As a result, some in-house data is going to be used for this step. All other experiments will use linear accelerometer and gyroscope data from IDNet gait dataset as input data.

For gait segmentation testing, performing data testing mentioned in section 4.3.1, constructing the set of metrics and performing differential testing proposed in extraction program mentioned in section 4.2.2 are the main focus. Specifically, for data testing, validating actual sampling frequency against the predefined sampling frequency set on the phone (which is 50Hz in our experiment), checking the order of timestamps collected from sensor data and validating x, y, and z data values are the procedure that will be conducted. Data is collected from various phones, including low-end, mid-end, and high-end devices. For extraction testing, the proposed metrics such as recall, alignment accuracy in section 4.2.2 will be constructed from IDNet gait dataset. To perform differential testings, two different candidate extraction programs will be used and the extraction will be done on the same dataset.

For gait classification, the testing experiments focus on measuring correctness, robustness, and efficiency of a trained model. Model architecture of the model used for testing is the architecture proposed in IDNet paper [1], and the dataset used for train and testing is IDNet gait dataset.

6 Experimental results

6.1 Gait segmentation testing

6.1.1 Data testing

Since IDNet dataset is pre-processed data, the data testing experiment used an in-house dataset in which sensors data was collected from various test users and devices during a long period of data collection. Throughout all the testings that were done, it is interesting to note that, some devices failed to send sensors data. It could be because of sensor hardware failures. However, for the devices that were able to send data, broken values such as *nan* value were rarely observed. Nevertheless,

the common issue that was observed is timestamps distribution of actual devices (shown in Figure 13 and 14). Specifically, Figure 13 and 14 show timestamps distribution of a problematic phone and a normal phone, respectively. For each figure, the blue line in the upper plot shows the actual timestamps collected from sample data, and the orange line shows the timestamps that the phone should have collected. The lower plot shows the distribution of time differences between two contiguous data points. Ideally, the time delta should be a constant number as it is equal to $1/sampling_rate$. And therefore, the expected distribution of time delta should be a straight line. It can be seen that even though the sampling frequency was defined in the code, the actual sampling frequency still varied. For some devices, the deviation of sampling frequency is relatively small (as shown in Figure 14), but for others, sampling frequency is non-uniform, and time delta between data points are sometimes significantly large or unsorted (as shown in Figure 13).

Depending on how the gait segmentation was designed and implemented, feeding noisy and non-uniform data might affect the performance of the extraction job. Therefore, if the extraction algorithm is dependent on a specific sampling frequency, additional steps such as sorting or data interpolation should be used to solve the issue.

6.1.2 Extraction program testing

Performing a test against ground truth data is a reliable way to measure the correctness of the program. However, given the expensiveness of the manual work on labelling data, the test does not scale. Therefore, having a static set with ground truth labels is a good but admittedly costly way to measure the correctness of the program. Within the scope of this thesis, the measurement of correctness (recall, precision) and alignment rate is demonstrated in a small session of walking data. The result of the session is illustrated in Figure 15. Specifically, from Figure 15, compared to the ground truth gait cycles, the extractor did not extract any false positive, and the extractor was missing 1 gait cycle (8 out of 9). Therefore, recall value is 8 gait cycles/9 gait cycles = 0.88, and precision value is 8 gait cycles/8 gait cycles = 1.0. Also, by applying formula 3, the average alignment rate is 0.971.

Other than correctness and alignment rate measurement, efficiency test does not require ground truth gait cycles data, and therefore, the testing was done on the entire IDNet gait dataset (135 sessions in total). Figure 16 shows the distribution of session length of all walking data collected in IDNet gait dataset (135 sessions) and the measured execution time per second of data on each session. It is worth noting that, the run-time metric collected from this test is not only useful to measure the efficiency of the program but can also give a good estimate of the latency introduced to the entire system coming from this extraction program.

Differential testing was also performed on IDNet gait dataset using two different candidate extraction programs. Figure 17 shows a high-level comparison between the number of gait cycles extracted from Extractor 1, Extractor 2, and the common

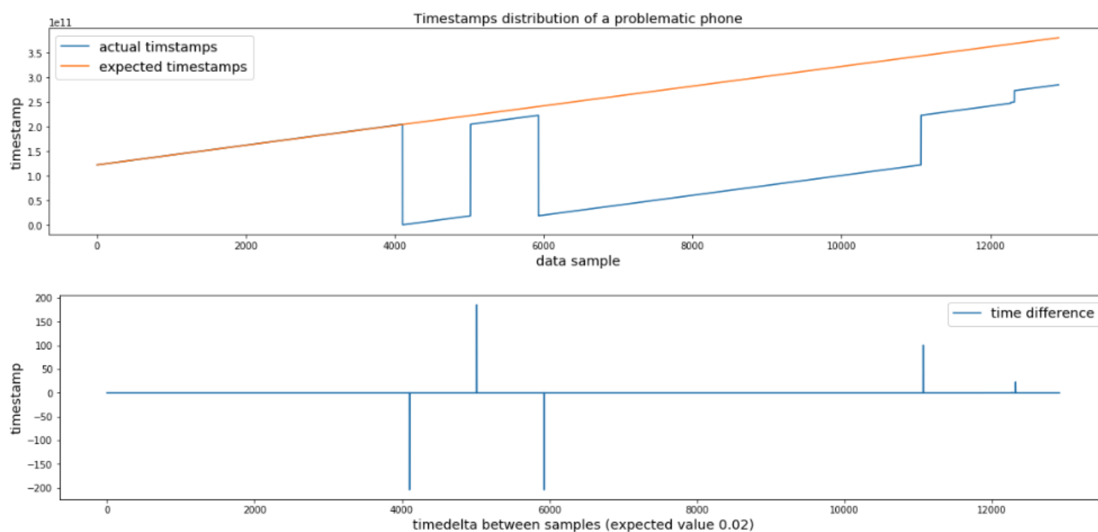


Figure 13: Timestamps of a walking session collected from a problematic device.

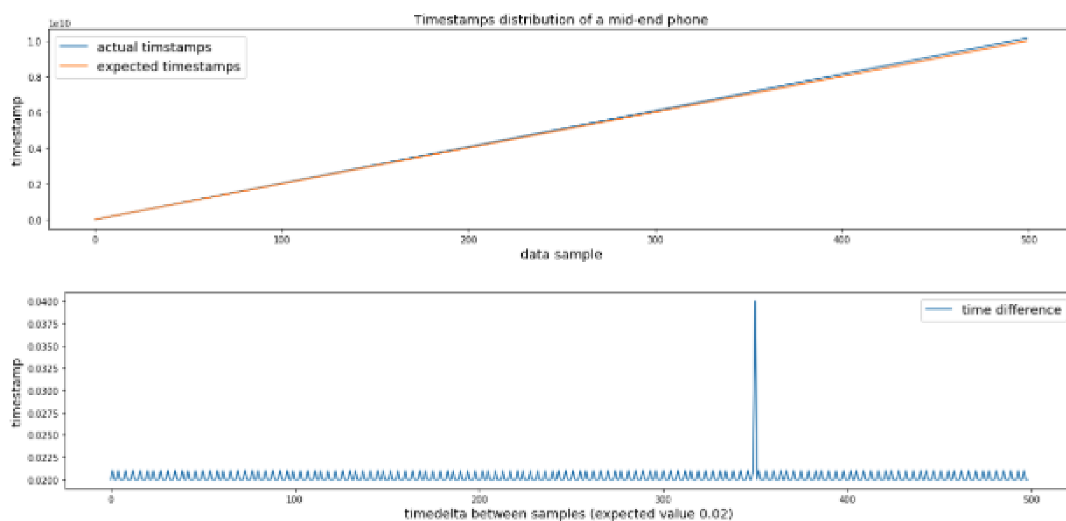


Figure 14: Timestamps of a walking session collected from a normal device.

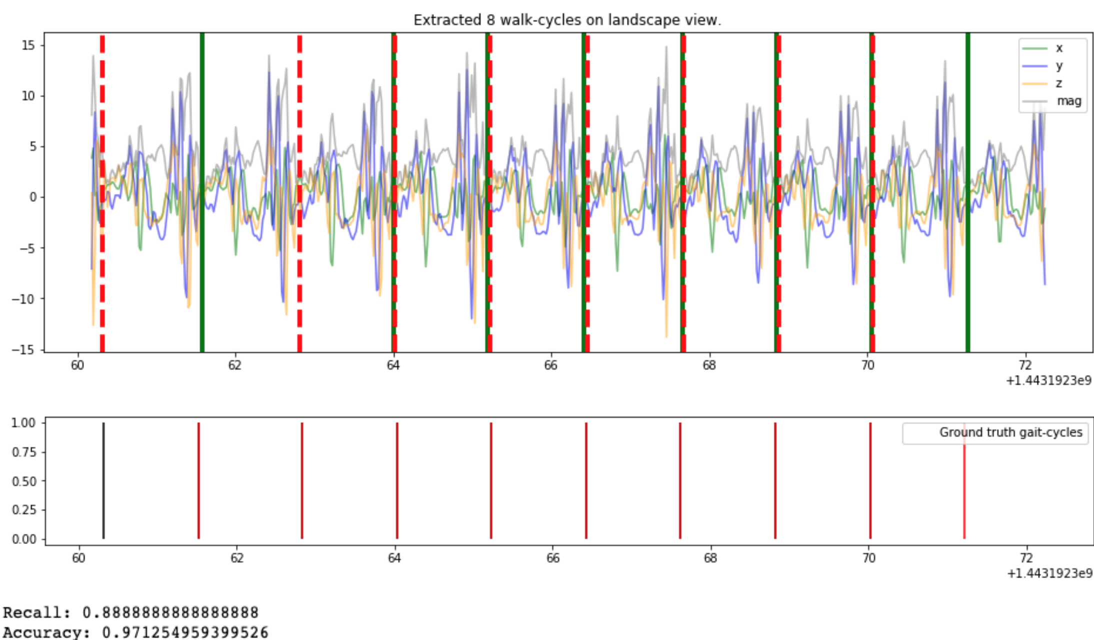


Figure 15: Extracted gait cycles from User 2, IDNet gait dataset (u002_w003_linearaccelerometer.log).
Note: the dotted red vertical lines mark the beginning of gait cycles and the solid green vertical lines mark the ending of gait cycles. Also, *mag* is an additional axis which is the magnitude value of x, y, and z axes.

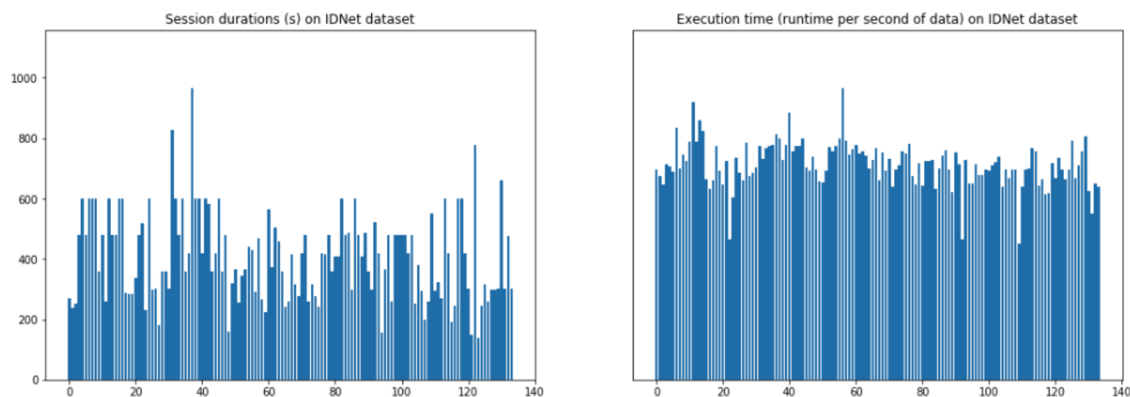


Figure 16: Sessions duration and execution time of gait segmentation on IDNet gait dataset.

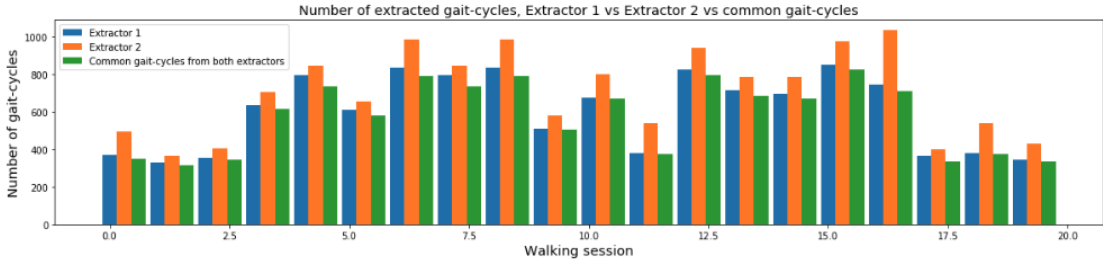


Figure 17: Comparison between the number of extracted gait cycles from Extractor 1, Extractor 2, and common gait cycles from both extractors.

gait cycles that were extracted from both extractors (for a better visualization, only the first 20 sessions in IDNet gait dataset was included in the plot). For a better understanding, common gait cycle is defined as a gait cycle that was extracted from both extractors. However, the extracted gait cycles from two different extractors might have a small time shift in start and end time depending on the way the algorithm was designed. Therefore, they are not expected to be identical (start and end time must not be identical). Instead, the below formula is used to identify if two candidate gait cycles from two different extractors are considered as common:

$$gc_1 = gc_2 \Leftrightarrow (gc_2.start_time \geq gc_1.start_time \;\&\; gc_2.start_time \leq gc_1.stop_time) \tag{4}$$

in which gc is a shorthand for gait cycle, gc_1 is a candidate gait cycle from extractor 1, and gc_2 is a candidate gait cycle from extractor 2.

By performing differential testing, Figure 17 gives an indication that Extractor 2 tends to extract more gait cycles than Extractor 1. And since the number of common gait cycles are relatively the same as the number of gait cycles extracted from Extractor 1, it can be seen as an indication that most of the extracted gait cycles extracted from Extractor 1 were also part of the results coming from Extractor 2. Last but not least, it turned out that most of the false positive gait cycles (gait cycles extracted from the extractor but are not actual gait cycles) extracted from the extractor fall into the uncommon set. For example, Figure 18 shows some examples of the false positives extracted from Extractor 2 found in the uncommon set between two extractors. By collecting and detecting false positives, an algorithm designer can keep improving on the robustness and correctness of the extraction program.

6.2 Gait classification testing

6.2.1 Data testing

Figure 19 shows the test results performed on 300 test users data from an in-house dataset. As can be seen from Figure 19, for most users, extracted gait cycles satisfied the predefined set of constraints presented in section 4.3.1. Nevertheless,

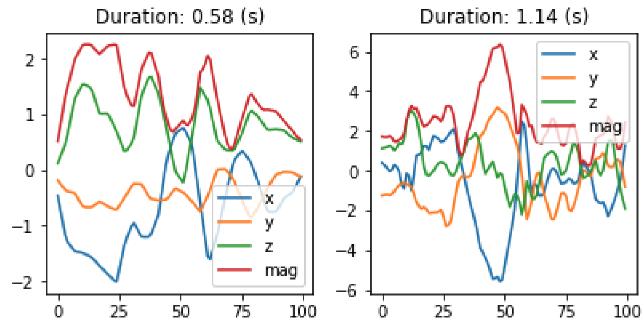


Figure 18: False positives from Extractor 2.
Note: although the above extracted candidates are high jerk signals, they do not represent walking patterns. Specifically, although the left and right plots are continuous signal (in time), their shapes do not look consistent (compared to the continuous walking data from Figure 15). Also, given that the time duration of the left candidate (0.58 seconds) is significantly smaller than the right candidate (1.14 seconds), it is an evidence that the extractor failed to extract some non-walking data.

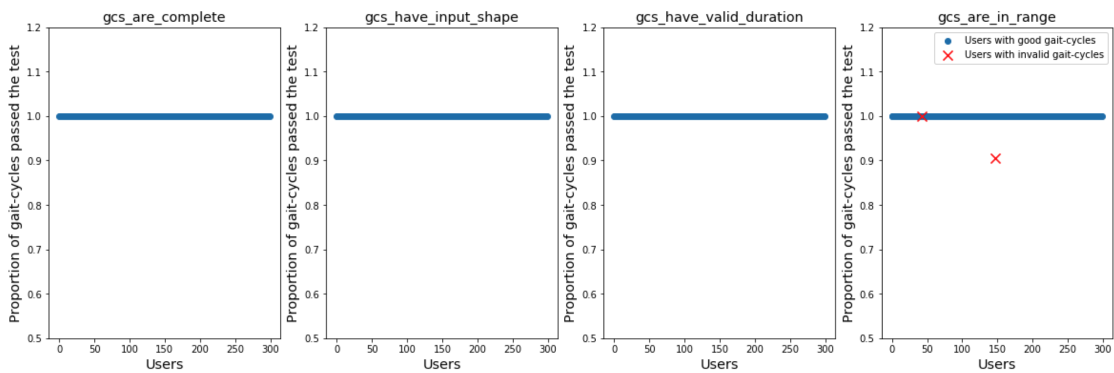


Figure 19: Results from four test cases (*gcs_are_complete*, *gcs_have_input_shape*, *gcs_have_valid_duration*, *gcs_are_in_range*) performed on 300 test users.

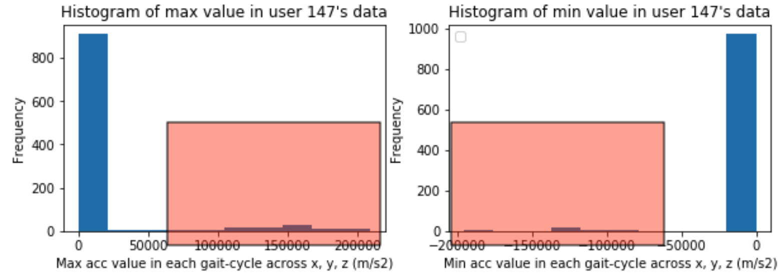


Figure 20: Distribution of acceleration value range across x, y, and z per gait cycle from user 147.

there is a minor set of users whose gait cycles did not qualify one of the constraints. Specifically, about 0.1% and 10% of gait cycles from user 42 and 147 had issues with the test case *gcs_are_in_range*, respectively. It is worth noting that, although these gait cycles passed the test *gcs_are_complete*, they still eventually failed the *gcs_are_in_range* test. It is because *gcs_are_complete* helps verifying that test data does not contain *nan* values, but *gcs_are_in_range* can help to verify that the values are in a reasonable range. Figure 20 shows the distribution of value range from gait cycles data coming from user 147 who was one of the users who failed at the test case *gcs_are_in_range*. As can be seen, most of the gait cycles' ranges fall into a reasonable range (both min and max). However, some gait cycles have values as large as 200000 m/s^2 and as small as -200000 m/s^2 (the highlighted boxes in Figure 20). These are inarguably problematic gait cycles since no human can walk with that range of acceleration. The root cause of the this might have derived from either invalid accelerometer data prior to the gait segmentation step, or the incorrect output produced by the extraction program.

Apart from applying a set of constraints to perform data testing, as mentioned in section 4.3.1, skew detection can also be a useful way to detect issues in users' data. Specifically, a trained model is supposed to perform well if and only if the training data and test data should come from the same data distribution and data schema should be identical. TensorFlow provides a framework for running skew detection on data, in both model training and model serving phases [37]. Although gait cycles data is not usable with TensorFlow validation framework, an in-house solution is developed based on the approach used by TensorFlow. Basically, there are three main types of skew that might occur in the data: *schema skew*, *feature skew*, and *distribution skew*. Schema skew detection [37] helps detecting some discrepancy between the features used in training and in serving. The reason this might happen is because usually training is done in a batch processing way, whereas model serving requires data streaming with a low latency. As a result, data pre-processing might be handled differently in two different codepaths, and the extracted features might be different from the two services [37]. In gait classification domain, since gait cycle is the only feature that the model requires, data schema can be considered as the shape of the gait cycles.

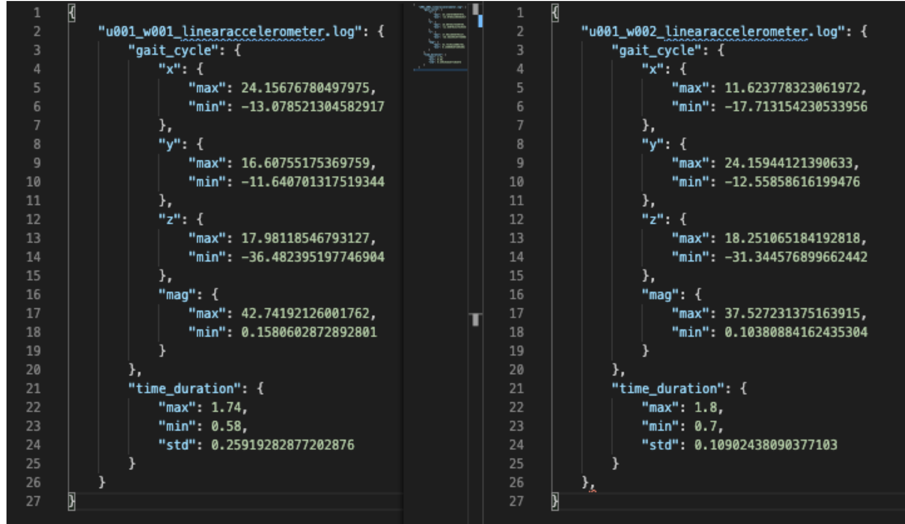


Figure 21: Features map generated from User 1 in two different walking sessions (IDNet gait dataset).

Unlike schema skew, constructing feature skew detection is done by constructing statistical features such as *min, max* in training data and use that to compare against serving or testing data. Also, feature skew detection will be constructed per user and this is different from person to person (in this case, it represents motion's feature values from a person). Figure 21 shows an example of statistical features maps constructed from user 1 in IDNet gait dataset in two different walking sessions. As can be seen from Figure 21, even though data comes from the same user, there is a noticeable difference in feature values in some axes (axis x and y) between two walking sessions. It is an evidence that human-gaits might have different walking modes, and therefore, the statistical feature values might look significantly different between the modes.

Last but not least, distribution skew detection is a way to combine all features in a statistical meaningful manner and compare that to the distribution of testing or serving data. In order to perform good predictions, the assumption is that training and the incoming test data are from the same distribution. There have been a lot of research and study about skew detection in text document or images [55, 56]; however, there have not been much of research focus on skew detection in gait classification, and therefore, it is not covered in the scope of this thesis. Nevertheless, it is worth noting that, there have been some prior works on skew detection in time series data [57, 58], and they can be seen as reference works for skew detection in gait classification.

6.2.2 Trained models testing

Model correctness. As described in section 4.3.4, for model correctness testing, True Acceptance Rate (TAR), False Rejection Rate (FRR), and ROC - AUC score

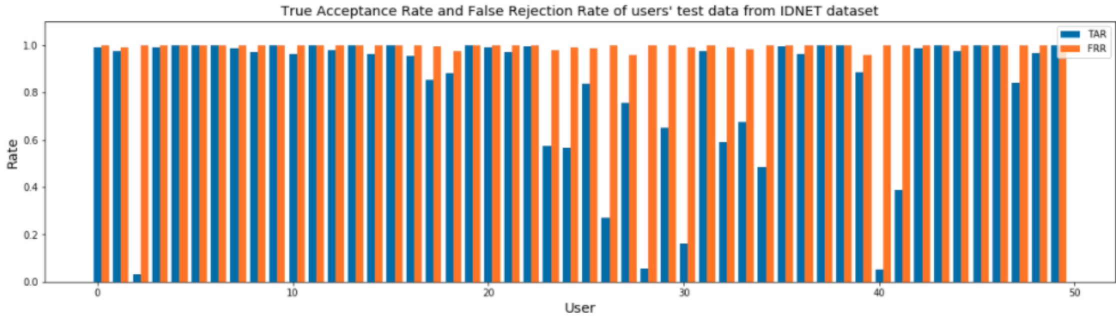


Figure 22: True Acceptance Rate and False Rejection Rate of 50 users in IDNet gait dataset with decision threshold at 0.5.
Note: It is worth noting that, the rates shown in the above figure are the results referenced to the decision threshold at 0.5. If the decision threshold changes, the classification results will look different.

will be computed for each user in the model. Since the model is multi-class classifier and the problem the model is solving is authentication problem (identify data as user or not user), the experiment setup is as follow: for each user, there are two different datasets that will be used for testing: *user data* and *non-users data* (data comes from the rest of the users). They are used to measure TAR and FRR respectively. For *user i*, for each given test point, the question being asked is: *"What is the probability of the given gait cycle belongs to user_i ?"*. Decision threshold used for this experiment is *0.5*. Put simply, if the probability is below 0.5, the decision is *"it is not user"*. Also, the value 0.5 is picked arbitrarily and just for the sake of this experiment. Figure 22 shows the distribution of TAR and FRR in IDNet gait dataset across all the users. As can be seen from the figure, the general trend is that FRR is consistently high, which means that the model will unlikely mispredict attackers' gait cycles as the true user's gait cycles. Nevertheless, for some users, TARs are noticeably low, which means that the model failed to predict the true user's data in such cases.

For the second experiment, ROC - AUC score is computed for each user. As can be seen from Figure 23, in overall, the ROC - AUC scores are relatively high, which means that the model is really good at distinguishing between true users and attackers' data. However, one noticeable observation is, that, ROC - AUC scores are extremely high across all users, including users who have low TAR in Figure 22. It is a strong indication that for these users, TARs could be improved significantly without compromising FRR had the decision threshold been picked carefully. For example, Figure 24 shows the progression of TPR , False Positive Rate (FPR) from User 2 based on the decision thresholds. As can be seen from the second plot in Figure 24, FPR does not change with the decision threshold between 1.75 and 0.1. On the one hand, if decision threshold is set as 0.5, TPR is significantly low (TPR = 0.03, FPR = 0.0). On the other hand, if decision threshold is set at around 0.1, TPR can reach its maximum without increasing FPR significantly (TPR = 1.0, FPR = 0.012). This observation can be an indication that having a good way of

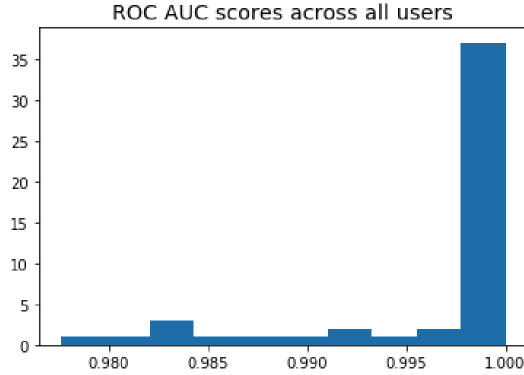


Figure 23: ROC - AUC scores across 50 users in IDNet gait dataset.

selecting decision threshold for each user can be one of the options which can help in improving model correctness. As an example, Figure 25 shows a remarkable improvement for TARs across all users when the decision threshold is set at 0.2 instead of 0.5 like in Figure 22.

Model robustness. Test data are generated using different strategies to test the robustness of the model. Specifically, three different strategies are being used: generate test data randomly using *random.rand()* function provided in the numpy library [60], generate test data with positive constant values in the range between 0 and 4 (reasonable acceleration value range), and with negative constant values between -4 and 0 using *arange()* function provided in the numpy library [61]. The motivation for doing this experiment is that these attacking scenarios are simple enough to perform in real world scenarios, and therefore, it can be used as a useful test to expose possible vulnerabilities before delivering the model to end users. In fact, Figure 26 shows that most of user classes perform strongly against synthesized attacking data. Nevertheless, some user classes really suffered from the tests. For instance, *user 4* has both TAR and FRR as *1.0* in both Figure 22 and 25. It is logical to make an assumption that the model is robust for this user. However, Figure 26 provided another perspective in which it shows *user 4* clearly suffered from the synthesized attacking data. The same issues also happened to *user 42*. It is worth noting that, these scenarios are simple to reproduce, and therefore, having solutions to prevent the misclassification in such cases is important, as it helps to avoid security loophole in the system.

Model efficiency. As described in section 4.3.4, inference time per sample can be used as the metric for model efficiency testing. Inference time is dependent on the hardware and environment setup of the test machines (contributing factors such as GPU vs CPU inference and software version). Therefore, in order to have a reliable test, it is necessary to have the same hardware and setup as the instances in which the production models will run. Also, having the metric is a useful way to estimate overhead latency that will be added to the system from this step. Figure 27 shows the average inference time on different batch sizes. The experiment was done on a personal computer; and therefore, the actual values do not matter. However, it

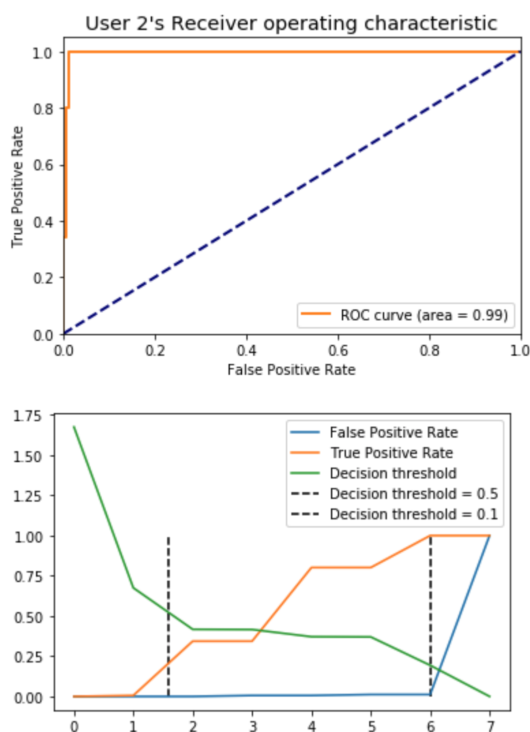


Figure 24: ROC curve and TPR, FPR, decision threshold curves of User 2 in IDNet gait dataset.
Note: for the lower subplot in the figure, x axis represents the number of instances where decision thresholds are being calculated. The size of x is equal to the length of test samples + 1 [59]. Also, y axis indicates the value of the rates and decision thresholds. It is worth noting that, the first decision threshold represents no instance being predicted, and the value is arbitrarily picked as $max(predicted_scores) + 1$ [59]

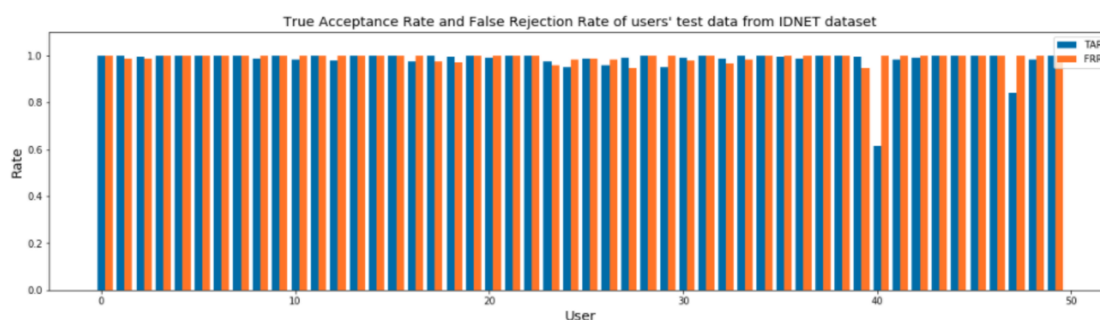


Figure 25: True Acceptance Rate and False Rejection Rate of 50 users in IDNet gait dataset with decision threshold at 0.2.

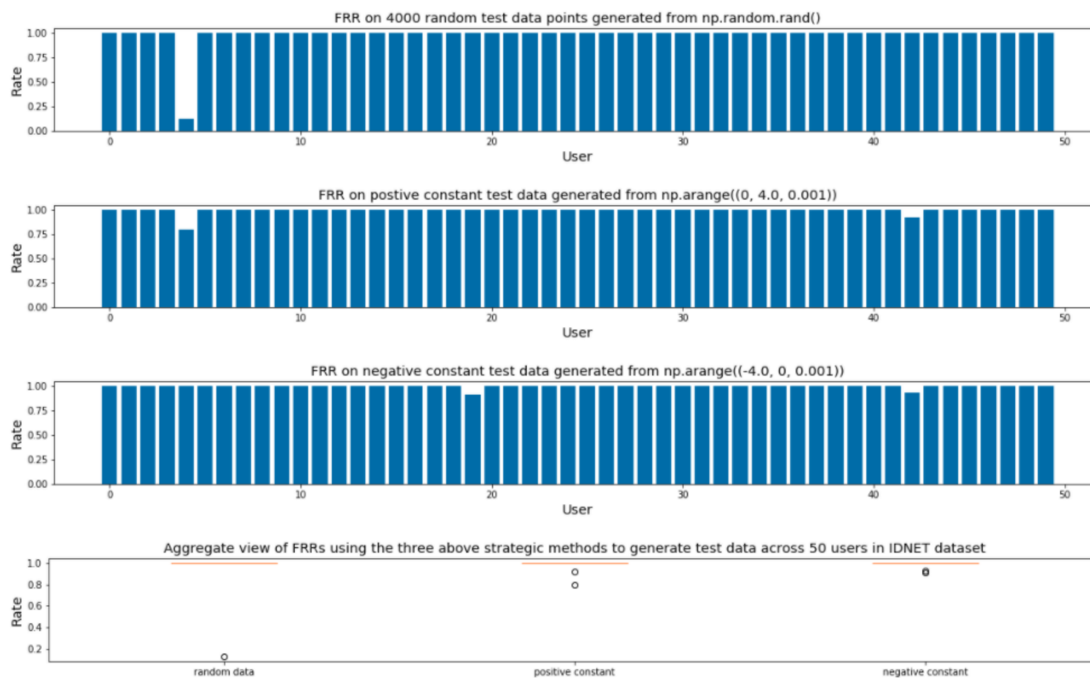


Figure 26: Distribution of FRRs across 50 users using different kinds of attacking data.

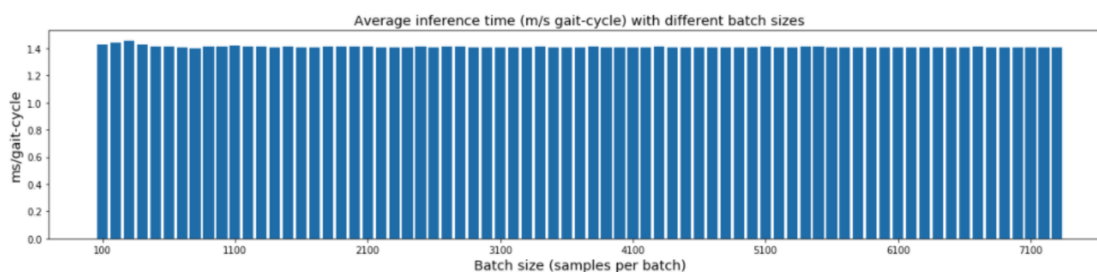


Figure 27: Average inference time per gait cycle on different batch sizes.

is worth noting that, the experiment is repeated 30 times to construct a fairest measurement, and the results shown in Figure 27 are the average results measured from these repeated runs. As can be seen from the plot, the inference time per gait cycle does not vary much on different batch sizes that were fed into the model.

7 Discussion

Experimental results presented in section 6 gives an evidence of the wide testing space that an ML system requires. There are many places where the system can go wrong, starting from raw input data, algorithm program, model architecture, or the outputs from each step. Based on the data testing experiment from section 4.2.1,

it can be seen that raw input data is sometimes unreliable, especially the sampling frequency of sensor data and the ordering of data points that were received from sensors. By observing this, an additional cleansing step is essential, and it will help to fix or filter broken input data from the early stages of the pipeline.

Section 4.2.1 introduced the metrics that can be used to test the correctness and efficiency of the program. Some experimental results were already introduced as a showcase on how the metric looks like while performing the test. Also, for a more exhaustive and scalable testing technique - differential testing was introduced in gait segmentation program testing. Basically, differential testing does not require human labeling on gait cycles, but it can help to verify possible issues of extraction programs by comparing the uncommon extracted gait cycles from the two candidates. Even though it might not be an ideal way to expose bugs since it could be the case that the two candidates might have the same issues, it is still a useful way to test the program since it is inexpensive to run, and the uncommon set can be tracked so that developers can re-visit in case some significant issues are noticed. Having several other candidate programs using different algorithms is also a way to help exposing bugs. Besides, measuring the extraction performance on negative cases is also important because extracting too much noise can affect the model accuracy, as it is a way to self-poison the model.

On gait classification data input testing, unit test-like approach was introduced in data testing, as it can be used as a way to validate the quality of each individual gait cycle based on a list of predefined constraints. It is similar to a list of test cases that developers can add in traditional software testing. More importantly, the list of constraints can evolve over time. Also, different skew detection techniques were mentioned and experimented in section 4.3.1 such as schema skew and features skew. They are useful tools to filter out obvious problematic data before feeding into the classifier. However, this thesis does not perform an actual experiment on a more statistical skew detection technique - distribution skew detection. In fact, distribution skew detection is a valuable way to detect the distribution shift between different datasets. Moreover, it is a good way to detect some abnormality of serving data compared to the data that user models were trained on. Specifically, since human-gait might change overtime due to different factors such as age, phone positions (e.g. in pocket, in hand, etc.), having distribution skew detection is a good way to tell the system it is time to retrain user models. Therefore, distribution skew detection on human-gait data is one of the interesting future works.

On trained models testing, different metrics inherited from biometric authentication testing such as TAR, FRR were introduced and measured on a per-user basis. They can be seen as a standard and useful way to look at model performance. Also, since solving user authentication is simply answering the question of user or not user, there has to be a way to convert from a soft value such as probability to a binary decision. Section 6.2.2 provided a look on ROC - AUC scores, which gives a better overview of the discriminability of the test model. Also, the analysis of Figure 24 gives a hint that optimizing decision threshold based on the TPR, FPR curves can possibly be a good way to perform model performance. Hence, this is also an

interesting future work that is worth investigating on. For robustness testing, some shallow testing experiments such as feeding random numbers or constant numbers with the correct data shape was done in section 6.2.2. Since it is not a sophisticated attack and this kind of data can be easily created, it raises an awareness that these scenarios should be well-tested before models deployment. One possible solution to prevent this kind of attacks is co-train users data with these artificial attacking data. It is worth noting that, although the actual experiment is not covered in this thesis, more sophisticated attacks such as data perturbation (adding small vibration into true user data) or synthesizing attackers data based on motion videos are also potentially good ways to test model robustness. Last but not least, inference time per gait cycle was also introduced as a metric to measure model efficiency. It is a useful statistic to look at, as it can give an estimation on how much latency it will contribute to the entire system.

Although section 6.2.2 presented different ways and metrics to test DL models, the testing technique used is black-box testing, in which the model is treated as an isolated component, and no actual interpretations or model understandings were required. As pointed out in section 4.3.4, performing white-box testing on gait DL models is one of the most interesting but challenging future works. It can help to understand more about a model's strengths and weaknesses, and the interconnections between the neurons in the model. Some suggested testing metrics such as neuron coverage might also be a good metric to start with [4]. Also, across the state-of-the-art white-box testing papers, one of the common techniques is to perform metamorphic testing, in which test data can be artificially created by changing things such as background color or weather condition while true labels remain unchanged [42, 4, 6]. It is non-trivial to apply the same technique in gait classification, since there is no such equivalent things in a human-gait domain. In addition, Figure 22 gives an indication that gait classifiers might work well for some users, but they might fail to classify some subset of users. It would be an interesting study to debug gait data as well as gait models from these users. Also, clustering gait data into different human activities could be a potential way to help in debugging user data.

More recently, tensorflow introduced Tensorflow Extended (TFX) [62] which has many useful features such as data validation, model analysis, and fairness indicator. Therefore, leveraging the production pipeline so that it can use these features will be a good way to utilize the functionalities provided by TFX. Besides, most of the proposed testing solutions such as performing correctness testing on extraction program, performing data testing using a set of predefined constrains, constructing data schema, and constructing models performance can be automated. And therefore, they can be implemented as a part of Continuous Integration (CI) pipeline in the production system. Other proposed testing solutions such as differential testing in gait segmentation extraction program or some future work such as model debugging might still require human validation.

8 Conclusion

Thanks to the current breakthrough of DL in computer vision, many other application domains have also tried to embrace DL as a viable candidate solution for solving challenging problems. As mentioned in section 2.2, there had been attempts in solving gait recognition using traditional ML techniques, but the common issues were the high cost of running and the hard-thinking features engineer required. Nevertheless, recent DL research on human-gait recognition on DL-based solutions suggested a new way for solving such challenging problems, especially when the solution does not require hand-crafted features engineer but they achieve significant improvements in terms of model accuracy and efficiency. Nevertheless, together with the wide adoption of DL, having a proper testing methodology and technique for the system is also an important part, as it is a way to ensure the reliability and quality of the system before deploying to large scale customers. The thesis gives an overview of authentication, biometric authentication, and how gait analysis can be utilized as one of the passive authentication factors. Furthermore, the thesis also gives an overview of the current state-of-the-art CNN-based gait classification techniques and solutions. Basically, there are two main common ML-based components that were used across this study: gait segmentation and gait-classifier. The thesis provides testing solutions for both components using a combination of matured traditional software development workflow - V model, and identifying the test space and characteristics that need to be tested in a gait authentication system. Throughout all the experiments, this study helps to point out possible problems and software issues that might occur in the system, starting from raw data coming from mobile devices, to DL model robustness testing on different testing scenarios. Last but not least, some interesting but challenging future work such as distribution skew detection in gait data, advance model debugging, or white-box testing in gait models have also been presented and discussed.

References

1 M. Gadaleta and M. Rossi, “Idnet: Smartphone-based gait recognition with convolutional neural networks,” *CoRR*, vol. abs/1606.03238, 2016.

2 S. Yao, S. Hu, Y. Zhao, A. Zhang, and T. F. Abdelzaher, “Deepsense: A unified deep learning framework for time-series mobile sensing data processing,” *CoRR*, vol. abs/1611.01942, 2016.

3 Ø. Stang, “Gait analysis: Is it easy to learn to walk like someone else?,” 2007.

4 K. Pei, Y. Cao, J. Yang, and S. Jana, “Deepxplore: Automated whitebox testing of deep learning systems,” *CoRR*, vol. abs/1705.06640, 2017.

5 H. B. Braiek and F. Khomh, “On testing machine learning programs,” *CoRR*, vol. abs/1812.02257, 2018.

6 J. M. Zhang, M. Harman, L. Ma, and Y. Liu, "Machine learning testing: Survey, landscapes and horizons," *CoRR*, vol. abs/1906.10742, 2019.

7 "V-model - javatpoint." `https://www.javatpoint.com/software-engineering-v-model`. Accessed: 2020-09-02.

8 C. Nickel, "Accelerometer-based biometric gait recognition for authentication on smartphones," 2012.

9 NuData, "Knowledge based authentication: Why kba is no longer a suitable method," Nov 2019.

10 M. Sultana, P. P. Paul, and M. Gavrilova, "A concept of social behavioral biometrics: Motivation, current developments, and future trends," in *2014 International Conference on Cyberworlds*, pp. 271–278, Oct 2014.

11 M. D. Marsico and A. Mecca, "A survey on gait recognition via wearable sensors," *ACM Comput. Surv.*, vol. 52, Aug. 2019.

12 "What is gait?." `https://study.com/academy/lesson/what-is-gait-definition-types-analysis-abnormalities.html`. Accessed: 2020-03-02.

13 C. Vaughan, B. Davis, and J. O'Connor, *Dynamics of human gait*. No. nid. 2 in Dynamics of Human Gait, Human Kinetics Publishers, 1992.

14 M. Derawi, "Accelerometer-based gait analysis, a survey," *Nor Informasjonssikkerhetskonferanse NISK*, 01 2010.

15 M. P. Mufandaidza, T. D. Ramotsoela, and G. P. Hancke, "Continuous user authentication in smartphones using gait analysis," in *IECON 2018 - 44th Annual Conference of the IEEE Industrial Electronics Society*, pp. 4656–4661, Oct 2018.

16 M. Muaaz and R. Mayrhofer, "Smartphone-based gait recognition: From authentication to imitation," *IEEE Transactions on Mobile Computing*, vol. 16, pp. 3209–3221, Nov 2017.

17 M. Muaaz and R. Mayrhofer, "An analysis of different approaches to gait recognition using cell phone based accelerometers," in *Proceedings of International Conference on Advances in Mobile Computing Multimedia*, MoMM '13, (New York, NY, USA), p. 293–300, Association for Computing Machinery, 2013.

18 "Smartphone gait signals - dataset, university of padova, italy." `http://signet.dei.unipd.it/wearables/IDNet_dataset.tar.gz`. Accessed: 2020-03-02.

19 "Sensors overview : Android developers." `https://developer.android.com/guide/topics/sensors/sensors_overview`. Accessed: 2020-10-02.

20 A. Graves, A. Mohamed, and G. E. Hinton, "Speech recognition with deep recurrent neural networks," *CoRR*, vol. abs/1303.5778, 2013.

21 Y. Jin, J. Li, D. Ma, X. Guo, and H. Yu, “A semi-automatic annotation technology for traffic scene image labeling based on deep learning preprocessing,” in *2017 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC)*, vol. 1, pp. 315–320, July 2017.

22 Q. Zou, Y. Wang, Y. Zhao, Q. Wang, C. Shen, and Q. Li, “Deep learning based gait recognition using smartphones in the wild,” *CoRR*, vol. abs/1811.00338, 2018.

23 Z. Wu, Y. Huang, L. Wang, X. Wang, and T. Tan, “A comprehensive study on cross-view gait based human identification with deep cnns,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, p. 209–226, Feb. 2017.

24 M. Zeng, L. T. Nguyen, B. Yu, O. J. Mengshoel, J. Zhu, P. Wu, and J. Zhang, “Convolutional neural networks for human activity recognition using mobile sensors,” in *6th International Conference on Mobile Computing, Applications and Services*, pp. 197–205, Nov 2014.

25 I. Papavasileiou, “An algorithmic framework for gait analysis and gait-based biometric authentication,” 2018.

26 H. Sanders and J. Saxe, “Garbage in, garbage out: How purportedly great ml models can be screwed up by bad data,”

27 L. Rong, D. Zhiguo, Z. Jianzhong, and L. Ming, “Identification of individual walking patterns using gait acceleration,” in *2007 1st International Conference on Bioinformatics and Biomedical Engineering*, pp. 543–546, 2007.

28 M. O. Derawi, P. Bours, and K. Holien, “Improved cycle detection for accelerometer based gait authentication,” in *2010 Sixth International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, pp. 312–317, 2010.

29 M. De Marsico and A. Mecca, “Biometric walk recognizer: Gait recognition by a single smartphone accelerometer,” *Multimedia Tools and Applications*, vol. 76, 06 2016.

30 M. D. Marsico and A. Mecca, “Biometric walk recognizer,” *Multimedia Tools and Applications*, vol. 76, pp. 4713–4745, Feb 2017.

31 A. Nambiar, A. Bernardino, and J. C. Nascimento, “Gait-based person reidentification: A survey,” *ACM Comput. Surv.*, vol. 52, Apr. 2019.

32 Z. Zhao, P. Zheng, S. Xu, and X. Wu, “Object detection with deep learning: A review,” *CoRR*, vol. abs/1807.05511, 2018.

33 “Overview of ml pipelines | testing and debugging in machine learning.” `https://developers.google.com/machine-learning/testing-debugging/pipeline/overview`. Accessed: 2020-01-02.

34 A. Bertolino, “Software testing research: Achievements, challenges, dreams,” in *Future of Software Engineering (FOSE ’07)*, pp. 85–103, May 2007.

35 P. Varhol, “How to test software in the age of machine learning.” https://techbeacon.com/enterprise-it/moving-targets-testing-software-age-machine-learning, Apr 2019. Accessed: 2020-01-02.

36 C. Murphy, G. Kaiser, and M. Arias, “An approach to software testing of machine learning applications,” pp. 167–, 01 2007.

37 “Tensorflow data validation: Checking and analyzing your data : Tfx.” https://www.tensorflow.org/tfx/guide/tfdv#training-serving_skew_detection_tfdv_training_serving_skew_detection. Accessed: 2020-03-02.

38 N. Hynes, D. Sculley, and M. Terry, “The data linter: Lightweight automated sanity checking for ml data sets,” 2017.

39 W. M. McKeeman, “Differential testing for software,” *DIGITAL TECHNICAL JOURNAL*, vol. 10, no. 1, pp. 100–107, 1998.

40 J. Brownlee, “What does stochastic mean in machine learning?.” https://machinelearningmastery.com/stochastic-in-machine-learning/, Nov 2019. Accessed: 2020-01-02.

41 L. Ma, F. Zhang, J. Sun, M. Xue, B. Li, F. Juefei-Xu, C. Xie, L. Li, Y. Liu, J. Zhao, and Y. Wang, “Deepmutation: Mutation testing of deep learning systems,” *CoRR*, vol. abs/1805.05206, 2018.

42 A. Odena and I. J. Goodfellow, “Tensorfuzz: Debugging neural networks with coverage-guided fuzzing,” in *ICML*, 2018.

43 D. Selsam, P. Liang, and D. L. Dill, “Developing bug-free machine learning systems with formal mathematics,” *CoRR*, vol. abs/1706.08605, 2017.

44 Q. Xiao, K. Li, D. Zhang, and W. Xu, “Security risks in deep learning implementations,” in *2018 IEEE Security and Privacy Workshops (SPW)*, pp. 123–128, May 2018.

45 Q. Guo, X. Xie, L. Ma, Q. Hu, R. Feng, L. Li, Y. Liu, J. Zhao, and X. Li, “An orchestrated empirical study on deep learning frameworks and platforms,” *CoRR*, vol. abs/1811.05187, 2018.

46 D. Cheng, C. Cao, C. Xu, and X. Ma, “Manifesting bugs in machine learning code: An explorative study with mutation testing,” in *2018 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, pp. 313–324, July 2018.

47 R. Werpachowski, A. György, and C. Szepesvári, “Detecting overfitting via adversarial examples,” *CoRR*, vol. abs/1903.02380, 2019.

48 F. Mohd-Yasin, N. Zaiyadi, D. J. Nagel, D. S. Ong, C. E. Korman, and A. R. Faidz, "Noise and reliability measurement of a three-axis micro-accelerometer," *Microelectron. Eng.*, vol. 86, p. 991–995, Apr. 2009.

49 "Sensors overview: Android developers." `https://developer.android.com/guide/topics/sensors/sensors_overview#sensors-monitor`. Accessed: 2020-10-02.

50 "Getting raw accelerometer events | apple developer documentation." `https://developer.apple.com/documentation/coremotion/getting_raw_accelerometer_events`. Accessed: 2020-10-02.

51 "Lstms for time series in pytorch." `https://www.jessicayung.com/lstms-for-time-series-in-pytorch/`, Sep 2018. Accessed: 2020-10.

52 "Time series forecasting : Tensorflow core." `https://www.tensorflow.org/tutorials/structured_data/time_series`. Accessed: 2020-10-02.

53 S. Schelter, D. Lange, P. Schmidt, M. Celikel, F. Biessmann, and A. Grafberger, "Automating large-scale data quality verification," *Proceedings of the VLDB Endowment*, vol. 11, p. 1781–1794, Jan 2018.

54 D. Thakkar, "Biometric performance metrics: Select the right solution." `https://www.bayometric.com/biometric-performance-metrics-select-right-solution/`, Aug 2018. Accessed: 2020-24-03.

55 A. Al-Khatatneh, S. A. Pitchay, and M. Al-qudah, "A review of skew detection techniques for document," in *2015 17th UKSim-AMSS International Conference on Modelling and Simulation (UKSim)*, pp. 316–321, March 2015.

56 P. Bezmaternykh and D. Nikolaev, "A document skew detection method using fast hough transform," in *Twelfth International Conference on Machine Vision (ICMV 2019)*, vol. 11433, p. 114330J, International Society for Optics and Photonics, 2020.

57 G. Ngunkeng and W. Ning, "Information approach for the change-point detection in the skew normal distribution and its applications," *Sequential Analysis*, vol. 33, no. 4, pp. 475–490, 2014.

58 M. Eling, "Fitting insurance claims to skewed distributions: Are the skew-normal and skew-student good models?," *Insurance: Mathematics and Economics*, vol. 51, no. 2, pp. 239–248, 2012.

59 "sklearn.metrics.roc_curve." `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve`. Accessed: 2020-04-02.

60 "numpy.random.rand." `https://docs.scipy.org/doc/numpy-1.14.0/reference/generated/numpy.random.rand.html`. Accessed: 2020-25-03.

61 "numpy.arange." `https://docs.scipy.org/doc/numpy/reference/generated/numpy.arange.html`. Accessed: 2020-25-03.

62 "Tensorflow extended (tfx): Ml production pipelines." `https://www.tensorflow.org/tfx`. Accessed: 2020-04-02.

63 J. T. Geiger, M. Hofmann, B. Schuller, and G. Rigoll, "Gait-based person identification by spectral, cepstral and energy-related audio features," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 458–462, May 2013.

64 H. V. Hoang and M. Tran, "Deepsense-inception: Gait identification from inertial sensors with inception-like architecture and recurrent network," in *2017 13th International Conference on Computational Intelligence and Security (CIS)*, pp. 594–598, Dec 2017.

65 E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, "The oracle problem in software testing: A survey," *IEEE Transactions on Software Engineering*, vol. 41, pp. 507–525, May 2015.

66 S. Masuda, K. Ono, T. Yasue, and N. Hosokawa, "A survey of software quality for machine learning applications," in *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pp. 279–284, April 2018.

67 "Online identification is getting more and more intrusive.." `https://www.economist.com/science-and-technology/2019/05/23/online-identification-is-getting-more-and-more-intrusive`. Accessed: 2020-25-03.