Using Synthetic Data to Model Mobile User Interface Interactions

Laoa Jalal

Faculty of Health, Science and Technology

Master thesis in Computer Science

Second Cycle, 30 hp (ECTS)

Supervisor: Assoc. Prof. Sebastian Herold, University of Karlstad, Karlstad, SWE

Examiner: Prof. Bestoun S. Ahmed, University of Karlstad, Karlstad, SWE

Karlstad, June 13th, 2023

Abstract

Usability testing within User Interface (UI) is a central part of assuring high-quality UI design that provides good user-experiences across multiple user-groups. The process of usability testing often times requires extensive collection of user feedback, preferably across multiple user groups, to ensure an unbiased observation of the potential design flaws within the UI design. Attaining feedback from certain user groups has shown to be challenging, due to factors such as medical conditions that limits the possibilities of users to participate in the usability test. An absence of these hard-to-access groups can lead to designs that fails to consider their unique needs and preferences, which may potentially result in a worse user experience for these individuals. In this thesis, we try to address the current gaps within data collection of usability tests by investigating whether the Generative Adversarial Network (GAN) framework can be used to generate high-quality synthetic user interactions of a particular UI gesture across multiple user groups. Moreover, a collection UI interaction of 2 user groups, namely the elderly and young population, was conducted where the UI interaction at focus was the drag-and-drop operation. The datasets, comprising of both user groups were trained on separate GANs, both using the doppelGANger architecture, and the generated synthetic data were evaluated based on its diversity, how well temporal correlations are preserved and its performance compared to the real data when used in a classification task. The experiment result shows that both GANs produces high-quality synthetic resemblances of the drag-and-drop operation, where the synthetic samples show both diversity and uniqueness when compared to the actual dataset. The synthetic dataset across both user groups also provides similar statistical properties within the original dataset, such as the per-sample length distribution and the temporal correlations within the sequences. Furthermore, the synthetic dataset shows, on average, similar performance achievements across precision, recall and F1 scores compared to the actual dataset when used to train a classifier to distinguish between the elderly and younger population drag-and-drop sequences. Further research regarding the use of multiple UI gestures, using a single GAN to generate UI interactions across multiple user groups, and performing a comparative study of different GAN architectures would provide valuable insights of unexplored potentials and possible limitations within this particular problem domain.

Keywords

Time Series Data, Generative Adversarial Networks, Synthetic Data Generator, Usability Testing, Machine Learning

Sammanfattning

Användbarhetstester inom Användargränssnitt (UI) är en central del för att säkerställa högkvalitativ UI-design som ger bra användarupplevelser över flera användargrupper. Processen med användbarhetstester kräver ofta omfattande insamling av användarfeedback, helst över flera användargrupper, för att säkerställa en opartisk observation av de potentiella designbristerna inom UI-designen. Att få feedback från vissa användargrupper har visat sig vara utmanande, på grund av faktorer som medicinska tillstånd som begränsar användarnas möjligheter att delta i användbarhetstestet. En frånvaro av dessa svåråtkomliga grupper kan leda till design som inte tar hänsyn till deras unika behov och preferenser, vilket potentiellt kan resultera i en sämre användarupplevelse för dessa individer. I den här avhandlingen försöker vi ta itu med de nuvarande brister inom datainsamling av användbarhetstester genom att undersöka om GAN-ramverket kan användas för att generera syntetiska användarinteraktioner av hög kvalitet av en viss UI-gest över flera användar grupper. Dessutom genomfördes en samling UI-interaktion av 2 användargrupper, nämligen den äldre och den unga befolkningen, där UI-interaktionen i fokus var dra-och-släpp-operationen. Datauppsättningarna, bestående av båda användargrupperna, tränades på separata GAN, båda med hjälp av doppelGANger-arkitekturen, och den genererade syntetiska data utvärderades baserat på dess mångfald, hur väl tidsmässiga korrelationer bevaras och dess prestanda jämfört med den verkliga data när de används i en klassificeringsuppgift. Experimentresultatet visar att båda GAN producerar högkvalitativa syntetiska likheter med dra-och-släpp-operationen, där de syntetiska proverna visar både mångfald och unikhet jämfört med den faktiska datamängden. Den syntetiska datamängden över båda användargrupperna ger också liknande statistiska egenskaper inom den ursprungliga datamängden, såsom längdfördelningen per prov och de tidsmässiga korrelationerna inom sekvenserna. Dessutom visar den syntetiska datamängden i genomsnitt liknande prestandaprestationer över precision, återkallelse och F1-poäng jämfört med den faktiska datamängden när den används för att träna en klassificerare för att skilja mellan dra-och-släpp-sekvenser för äldre och yngre. Ytterligare forskning angående användningen av flera UI-gester, att använda en enda GAN för att generera UI-interaktioner över flera användargrupper och att utföra en jämförande studie av olika GAN-arkitekturer skulle ge värdefulla insikter om outforskade potentialer och möjliga begränsningar inom just denna problemdomän.

Nyckelord

Tidsseriedata, Generativa Motstridande Nätverk, Syntetisk datagenerator, Användbarhetstestning, Maskininlärning

Acknowledgements

First, I would like to express my gratitude to my supervisor, Sebastian Herold, for granting me valuable guidance, both in the writing process, as well as in the research as whole. Furthermore, I would also want to thank Bilal Maqbool for helping me with the collection of user data. Last but not least, I want to give a special thanks to both my sisters for their endless support in not only my academic journey, but also through the though times in life.

Acronyms

SDG	Synthetic Data Generator
GAN	Generative Adversarial Network
RNN	Recurrent Neural Network
LSTM	Long Short-Term Memory
UI	User Interface
CSV	Comma-separated Values
MSE	Mean Squared Error
BCE	Binary Cross-Entropy
GDPR	General Data Protection Regulation
UUID	Universally Unique IDentifier

Contents

Chapter 1

Introduction

1.1 Background

Today’s society heavily depends on complex software systems to operate as intended. Many of these software systems are deployed on mobile devices, often providing a UI that enables the end user to interact with the underlining system. The number of smartphone users has rapidly increased in the past decade, where the worldwide estimate of 2023 amounts to 5.2 billion users, almost a 370% increase compared to 2013 [10]. The number of smartphone users is expected to increase further, which opens new challenges for developers to provide high-quality services and adapt the service design for a wide range of user groups. Designing a user-friendly UI for smartphone devices is often considered more challenging than for desktop UIs, mainly because of the limited screen sizes and the unique set of UI gestures present for a smartphone[4]. This has led to various design guidelines provided by different platforms to help developers design a UI that offers a high-quality user experience [29] [17]. Even if the UI follows various design guidelines, testing the design on real users is often required for assurance regarding the user experience. These types of tests are referred to as usability testing and aims to detect possible design flaws within the current design by collecting user feedback of users that performs a set of specified tasks within the UI[42]. This process often requires multiple participants, preferably a wide range of user groups, to provide an unbiased observation of the potential design flaws. Usability testing of various kinds has been a widely studied topic, where some analyze the effect of different UI gestures across multiple user groups [23] [38], while others provide a general methodology for efficient usability testing [31].

1.2 Problem Statement

Usability testing is a crucial aspect of UI design, aiming to ensure satisfactory user experiences across diverse user groups. However, gathering usability data from certain groups presents significant challenges due to factors such as medical conditions, which can create both practical and ethical problems during the data collection process. The inability to effectively include these hard-to-access groups in usability testing can lead to designs that fail to consider their unique needs and preferences, potentially resulting in worse user experiences for these individuals. There are, however, limited studies regarding the augmentation of existing data gathered from usability testing. Augmentation of the limited data helps minimize the potential bias across various user groups when performing the usability testing, ultimately assisting the developers in designing a user-friendly UI adapted for all user types.

1.3 Thesis Objective

The aim of this project is to investigate whether a Synthetic Data Generator (SDG) can be used to generate high-quality user interaction data of a certain UI gesture from various user groups. This will be done by first capturing user interactions of various user groups and use the collected data to train a SDG using the GAN framework. The resulting SDG will be evaluated according to its diversity in generated UI gestures, how well the data resembles actual user interactions and its performance compared to the real data when used in a classification problem.

1.4 Goals

The thesis aims to combine the gaps between the use of GANs in the domain of usability testing, where the focus is on user interactions within the UI. More specifically, we will focus on a specific UI gesture, namely the drag and drop operation which is commonly used in many applications. The goal is to provide valuable insight, possibilities, and limitations of using GANs to augment the drag and drop operation of various user groups. The project is designed to be a stepping stone in motivating further research within this area of study to explore more complex settings where multiple UI gestures may be explored. Furthermore, the following research questions will be used to achieve the goals of the study:

- **RQ1**: How well does synthetic data resemble UI interaction data for different user groups?
- **RQ2**: How well does the synthetic UI interaction perform compared to real user interaction data?

1.5 Ethics and Sustainability

AI solutions have been greatly beneficial within many domains where prior solutions could be more efficient, impractical, or both. Many AI solutions, such as GANs, do, however, pose issues regarding bias and discrimination. An AI system trained on biased data can pose threats regarding its perception of right and wrong, which might pose disadvantages for certain demographic groups[19]. In our problem setting, the resulting synthetic data produced by our generators may not correctly represent the user group's actual behavior when interacting with the UI, which may create a false perception of how the user groups actually behave. The problem can be relatively serious if one mindlessly uses synthetic data without evaluating its quality. As mentioned earlier, one can limit the consequence of the problem by performing a statistical assessment of the synthetic data and comparing it against the actual data before using the synthetic data in a real problem setting. In conjunction, the thesis also requires user data for training the GANs. User data collection must comply with the General Data Protection Regulation (GDPR), which may otherwise lead to infringement [8]. All user data is stored anonymously to mitigate the possibility of breaking any rules within GDPR.

1.6 Methodology

The thesis is divided into two parts, the data collection phase and the experimental stage. The data is collected by implementing a prototype mobile application that captures the UI interaction alongside certain demographic information and stores it persistently. The collection of data is done by approaching possible participants that are willing to perform the experiment. After completing the data collection, the data is separated based on the user groups and pre-processed before training the GAN model. After training, the GAN is first visually evaluated based on its outputs, and depending on the visual assessment, we either change parameters within our GAN or use the GAN for a formal evaluation. The formal evaluation looks at statistical similarities between the real and synthetic user interaction and the difference in performance when used to train a classification model.

1.7 Delimitations

The thesis focuses primarily on a single type of UI interaction, the drag-and-drop operation, which is one of many UI gestures commonly used within a smartphone. Including a broader set of UI gestures will provide more insight into the strengths and weaknesses of our current approach and also generalize the proposed solution. The study focuses mainly on a single GAN architecture which may not provide the best solution for our problem settings.

1.8 Outline

The thesis contains the following chapters: Chapter 2 presents the necessary theory that is needed to understand the upcoming chapters. Alongside the theory, we also provide a section of related work that present similar studies to our research. Chapter 3 contains the various design decisions which are aimed to answer the research questions. Chapter 4 provides the practical implementation of the design decisions made from previous chapter. Chapter 5 presents the results of our study together with an discussion regarding the gathered results. Lastly, a conclusion is drawn in chapter 6 to sum up the thesis.

Chapter 2

Background and Related Work

This chapter provides both taxonomy and theory of various topics which are necessary to understand before reading upcoming chapters. We start with a walkthrough of the common UI gestures within a smartphone, alongside a brief explanation of UI testing. Then, a comprehensive introduction to neural network, alongside recurrent neural network is discussed which is later followed by the GAN framework and its theory. Lastly, we discuss related studies with the focus on usability testing and GANs adapted for time series data.

2.1 UI Interactions

Modern-day smartphones often provide a touchscreen for the users to interact with the various application within the phone. Touchscreen devices provide a diverse set of gestures to ease the process of various actions within the smartphone. The simplest gestures, often referred to as *Single-Touch Gestures*, involve actions using a single touch point [40]. These types of gestures involve actions such as tapping a button, displacement of elements using dragging, and swiping to scroll the UI, to name a few. Another set of gestures called *Multi-Touch Gestures*, requires the user to use multiple touch points to perform the desired operation [40]. An example of a common *Multi-Touch Gesture* is the pinch operation, commonly used to resize elements within the UI, such as pictures. Additionally, some smartphones include built-in sensor devices that analyze the phone's external surroundings, often referred to as *Sensor-based Gestures* [34]. A commonly used sensor device is the accelerometer which reports a quantitative acceleration value along the device's three axes [28].

How various user groups interact with the UI can vary based on factors such as age, physical abilities, and cultural background. For instance, people with visual impairment require special assistive technologies where the user receives spoken feedback when interacting with various UI components within the smartphone [26]. Alongside the physical limitations, younger users generally tend to navigate through UIs faster than the elderly population, where the elderly often navigates the UI slower and requires more explicit descriptions to navigate across the interface [32, 38]. Cultural background is also essential to consider, as some cultures interpret various interactions differently, e.g., different interpretations of symbols and color symbolism, which may affect how the users interact with the UI.

All the factors mentioned above must be considered when designing a UI to maximize the product's desired functionality and provide a high level of user experience. To ensure the UI is satisfactory, we often must perform various UI tests for complete quality assurance. UI testing contains multiple tasks, each with its purpose and goal of achieving high-quality design. These tasks range from simple design testing, where we often follow a set of guidelines to ensure consistency in layouts, colors, and font sizes, to name a few [29] [17]. Another important aspect of UI testing, referred to as interaction testing, ensures that the user can interact with the UI in the way it is intended for [21]. Lastly, to ensure that the UI design is satisfactory, we provide the UI for actual users to operate on various tasks, ultimately providing feedback on the various usability aspects to ensure whether the UI is user-friendly or not.

This thesis focuses primarily on the usability testing part of UI testing, where the main task is the drag-and-drop gesture, which requires the user to displace an element to a designated location.

2.2 Neural Networks

Neural networks, or Artificial neural networks (ANN), are a sub-domain of machine learning that has gained much attention in the past decades due to their promising results in various domains, such as speech recognition and natural language processing, to name a few [24]. Using a mathematical model, neural networks were designed to mimic the biological neurons inside the animal brain. Each artificial neuron consists of inputs that are processed inside the neuron to produce a single output. The output can be sent to multiple neurons, which take the previous neuron's output as its input. When multiple neurons are connected in sequence, with each output connecting to another neuron's input, we say that the network consists of multiple layers. The general goal of using a neural network is to approximate some function $g(\mathbf{x};\theta)$, where an input x, parameterized by θ, is used to map some output y. These models are often referred to as *feedforward* networks as the input $\mathbf{x}$ is processed sequentially from each layer until reaching the final output layer [12].

Figure 2.2.1 demonstrates a feedforward neural network with three layers, the mandatory input and output layers and a single hidden layer. When observing the figure, we see that each feature from the input layer is connected to each of the units inside the hidden layer. The edges connecting each neuron carry an implicit weight that determines the magnitude of each input. More formally, the neurons can be described

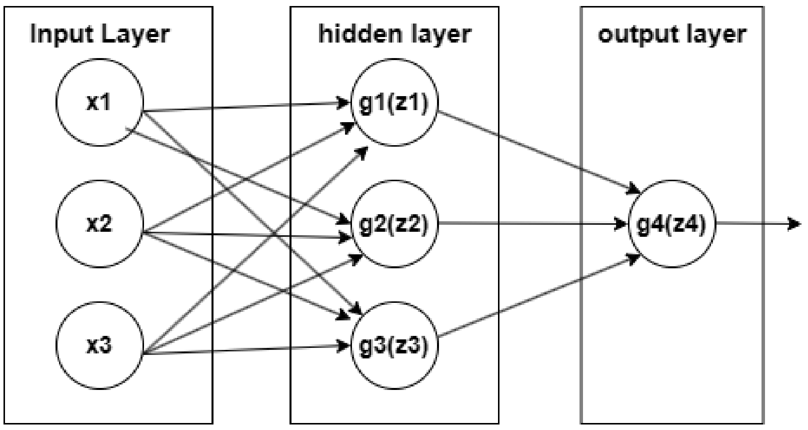


Figure 2.2.1: Neural network with an input layer consisting of 3 features, a single hidden layer with 3 units and a single unit in the output layer

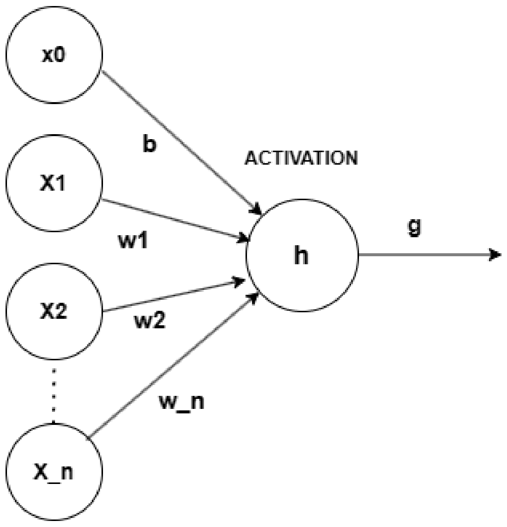


Figure 2.2.2: Single Neuron with n inputs which produce the output value using the activation function

with the following expression,

$$g_l^k = h(\sum_{i=1}^{n} w_{li}^{k-1} x_i + b_l^{k-1}) \tag{2.1}$$

To decode the expression in 2.1, we start by looking at the inner term containing the sum. The term $w_{li}^{k-1}x_i$ says that each in-bounding edge, which we refer to as weights w_{li}^{k-1}, are multiplied with the actual input value, x_i, where i refers to the connected edges, l the current activation unit and k the current layer. The term b_l^{k-1} is the bias we use to shift the function, similar to how the constant in a linear equation shifts the function in the y-axis. Figure 2.2.2 depicts an illustration of 2.1.

Summing all terms leaves a scalar value inside the activation function h. The activation function used inside the neuron is almost always non-linear. The non-linearity enables the neuron network to generalize to a broader family of functions, enabling more sophisticated models that can learn more complex patterns in contrast to using exclusively linear functions [24].

There are many available activation functions, each with its benefits and limitations. A commonly used activation function is the rectified linear unit **ReLU**, which uses the activation function $h(z) = max\{0, z\}$, where z is the linear summation of the input to the neuron. The ReLU function is similar to the linear function, except that negative values are crunched to 0. Despite ReLU's simplicity, it has shown great results in many architectures and is often used as the standard activation unit inside many hidden layers [1]. Other popular activation functions are the sigmoid function,

$$h(z) = \frac{1}{1 + e^{-z}} \tag{2.2}$$

and the hyperbolic tangent, $tanh$.

$$h(z) = tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \tag{2.3}$$

Both these activation functions are not preferred in a feed-forward model as the function saturates between (0,1) for the sigmoid and (-1, 1) for $tanh$ when the input z is too large or too small. Note that the z value is a linear summation of the input values multiplied by the input weights. The problem with saturating activation functions is the vanishing gradient problem which occurs when the derivative approaches 0. A gradient close to zero negatively affects the back-propagation algorithm, which uses gradient descent to update each parameter inside the neural network to minimize the loss value. Despite the drawbacks of using sigmoid and $tanh$ as activation functions inside the hidden layers for a feed-forward network, they are still useful in other settings, such as in Recurrent Neural Network (RNN).

After the input x reaches the output layer, we say the network has made a single forward propagation. To train the model, the output produced by the neural network has to be compared with some reference value y_{real} to properly adjust each weight inside the neural network. The calculated difference between the reference point and the model output is often called the loss value. There are several ways to calculate the loss value of a models prediction. One common loss function is the Mean Squared Error (MSE),

$$MSE = \frac{1}{N} \sum (y_{real} - y_{model})^2 \tag{2.4}$$

which takes the average sum of the squared distance between the real answer and the answer produced by the model. The MSE loss is commonly used when the model is trained on regression problems where the value ranges in $y \in \mathbb{R}^n$, where n is the number of output nodes in the network [18]. Another common loss function is the cross-entropy loss which is commonly used in problems where the output is of categorical nature [18]. For instance, we might have a dataset **X** containing images of cats and dogs with a corresponding target space $\mathbf{y} \in [cat, dog]$. In the case of classifying cats from dogs, we say that the problem is of binary nature where the output only ranges between 2 values. The loss function for the binary cross-entropy is defined as

$$BCE = -\frac{1}{N}\sum(y_{target} * log(y_{pred}) + (1 - y_{target}) * log(1 - y_{pred})) \tag{2.5}$$

where y_{target} is the actual label to the input, and y_{pred} is the models prediction, given the input. The logic behind cross-entropy is that, when y_{target} is 1 and the model predicts a value close to 0, the term $log(y_{pred}) \to \infty$, meaning that the loss is large when the model makes bad predictions. On the contrary, if the value predicted by the model is close to y_{target}, the loss is close to 0.

2.2.1 Recurrent Neural Networks

Recurrent Neural Networks, also called RNNs, are neural networks specializing in processing sequential data [12, 35]. The motivation behind RNNs was that the traditional feed-forward neural network, discussed in the previous section, was not designed to handle inputs with temporal correlation, such as text or time-series sequences where the input feature $\mathbf{x_i}$ depends on the previous feature $\mathbf{x_{i-1}}$. RNNs use recurrent states, often denoted as h, as additional input alongside the input x. More formally, the state at point t can be written with the following recursive equation,

$$h^t = g(W * h^{t-1}, V * x^t; \theta) \tag{2.6}$$

where h^t is the current state, based on the output of the activation function with input data x^t multiplied with the weight matrix **V**, alongside the previous state h^{t-1} multiplied by the weight matrix **W**. Figure 2.2.3 shows an example RNN with the many-to-many setting, meaning that each input maps to a corresponding output. Also, note that each layer shares the same weight matrices $\mathbf{A}, \mathbf{B}$, and **W**, meaning that each timestep contributes to adjusting the global weight when optimizing the network. RNNs are inherently sequential, where each hidden state h^t depends on the computation of the previous hidden state h^{t-1}. This makes it inherently hard to utilize parallel computing, making RNNs very slow compared to other architectures [35].

Alongside being slow to train, RNNs are also very hard to train. When applying the backpropagation algorithm to an RNN, we often say we backpropagate through time. Because each hidden state h^t depends on the previous states $h^1..h^{t-1}$, we have to consider the loss gradients with respect to each prior state. To calculate the loss of state T with respect to the weight matrix **W**, we can use the following equation,

$$\frac{\partial L_T}{\partial W} = \sum_{i=1}^{T} \frac{\partial L_T}{\partial y_T} \frac{\partial y_T}{\partial h_i} \frac{\partial h_i}{\partial W} \tag{2.7}$$

When observing 2.7, the gradient calculation accumulates rapidly to either zero or infinity when the RNN contains long sequences because of the repeating multiplication of the gradients. One proposed solution for handling the vanishing gradient problem is to use Long Short-Term Memory (LSTM) [35]. LSTM is an modified RNN cell that utilize additional gates to reduce the risk of vanishing gradients during the backpropagation.

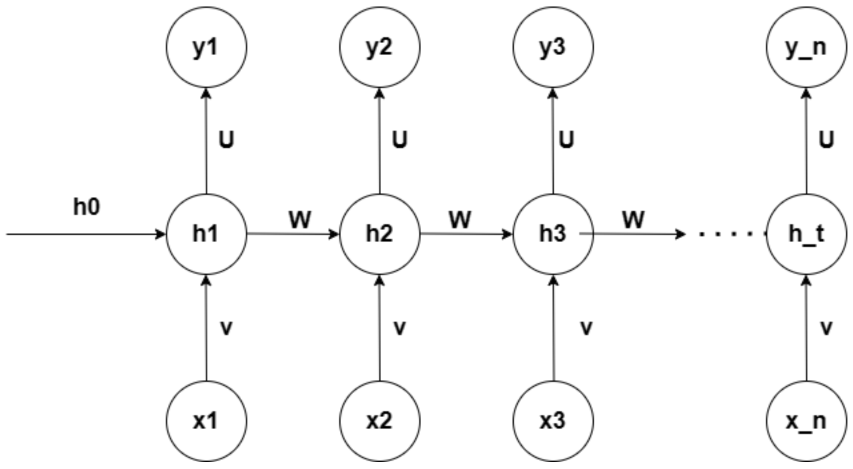


Figure 2.2.3: A unfolded RNN where each input maps to a corresponding output. This is often referred to as a many-to-many architecture

Long Short-Term Memory

The LSTM unit, first proposed in [16], was designed to address the problems within the vanilla RNN unit, where long-term dependencies often led to vanishing gradient problems when training the RNN. The LSTM unit introduces a new input channel called the cell state c^t at each hidden unit; see figure 2.2.4. The cell uses three unique gates to control the information traversing the cell state: the input, output and forget gates. As with the normal RNN cell, we can describe the LSTM cell mathematically using the following set of equations,

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \tag{2.8}$$
$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \tag{2.9}$$
$$\tilde{C_t} = tanh(W_c[h_{t-1}, x_t] + b_c) \tag{2.10}$$
$$C_t = f_t * C_{t-1} + i_t * \tilde{C_t} \tag{2.11}$$
$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{2.12}$$
$$h_t = o_t * tanh(C^t) \tag{2.13}$$

The equation 2.8 is the forget gate which uses sigmoid activation to crunch the previous state h_{t-1} and current input x_t to a value between 0 and 1. If the gate outputs 0, we forget the previous cell state C_{t-1} because $C_{t-1} * 0 = 0$. On the contrary, if the gate outputs 1, all information in C_{t-1} is used in the current cell state. Values between 0 and 1 scale the previous cell state according to the relevance of the current state. To store and update new information to the current cell state C_t, we use the equations 2.9, 2.10 and 2.11. The equation 2.9 determines which values to update from the input, while 2.10 generates values that determine the magnitude each value should either increase or decrease with. To update the current cell, the values produced by i_t and $\tilde{C}_t$ are multiplied and added with the previous cell state, and scaled with f_t. The final operation happens in the output gate, where the hidden state h_t is calculated by first determining the magnitude of the output, which is multiplied by $tanh(C^t)$.

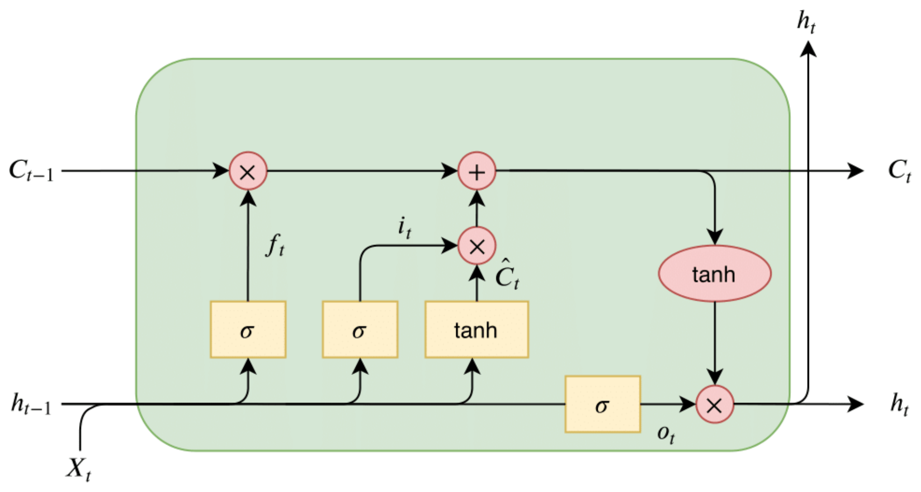


Figure 2.2.4: Diagram over LSTM cell

2.3 Generative models

In machine learning, two distinct classes are used to solve a set of problems, namely the discriminative and generative models. Discriminative models are used in problems where we try to find the probability of an outcome **y**, given the input **x**, which can be formulated using the conditional probability $p(y|x)$. This class of models are often used in settings where we have labeled data and wish to learn the underlying pattern of a dataset to distinguish between different targets. One example is training a model to distinguish between cats and dogs, where the input **x** may be an image containing either a cat or dog, which we mark with a label **y**. When we want to learn the underlying probability distribution of a given dataset X, given some conditional description Y we need generative models that tries to learn the conditional probability of $p(X|Y)$. To connect the example with the cats and dogs, we may train a generative model to generate cat and dog images, X, given an arbitrary input description Y. Generative models are often both harder and slower to train than the discriminative models. While the discriminative models learns to only draw a decision boundary for its given data space, generative models has to learn the underlying distribution of the given dataset [15]. Various generative classes of models exist, each with its benefit and limitation. In this thesis, GANs are used as the method for training a model to learn the underlying distribution of a given dataset.

2.4 Generative Adversarial Networks

GANs is a framework, first proposed by [13], for training a generative model by introducing two separate models, the generator G and the discriminator D. GANs use the principle of game theory where the two models, D and G, compete in a zero-sum game, meaning that one actor's gain results in the other actor's loss. Figure 2.4.1 shows a generic GAN network's architectural components and information flow. The generator G takes a random noise vector z from an explicitly defined distribution, often in the form of a Gaussian or uniform distribution, as input and produces the output $G(z)$. The discriminator D is trained to classify whether the incoming data is real or fake. Real data refers to the actual sample domain, that is, the dataset x_{real} while fake data is generated by $G(z)$. The discriminator is essentially a binary classifier where a value of 0 indicates fake samples and 1 is real samples. The ultimate goal of the generator is to fool the discriminator such that the probability of $D(G(z))$ is close to 0.5, meaning that the discriminator essentially has to guess whether the input is either fake or real. To achieve the aforementioned goal, the generator has to adapt according to the loss produced by the discriminator. Recall that the loss is generally calculated by measuring the distance between the desired output and the actual generated output with some explicitly defined loss function. To understand the loss function used in GAN, we have to revisit the Binary Cross-Entropy (BCE) loss function 2.5, which we refactor according to our discriminator D and the output of $G(z)$, which gives us the following equation:

$$\min_{G} \max_{D} \mathbb{E}_{x\sim p_{data}(x)}[log(D(x))] + \mathbb{E}_{z\sim p_z(z)}[log(1 - D(G(z))] \tag{2.14}$$

It has been shown that the optimal solution always occurs when $p_{data}(x) = p_z(z)$ [12]. Because the optimal solution always occurs when both distributions are equal, solving the Jensen-Shannon divergence between $p_{data}(x)$ and $p_z(z)$ is equivalent to solving the optimization in 2.14.

Although if the loss function in 2.14 has been proven to eventually reach the optimal solution when trained, the practical results have shown that the models tend to converge very slowly and also suffer from saturated gradients, which eventually halts the process of gradient-based optimization. To improve the training process, a modification of 2.14 was proposed by [13], which shows that one still reaches the exact optimal solution if $\mathbb{E}_{z\sim p_z(z)}[log(1 - D(G(z))]$ is changed to $-\mathbb{E}_{z\sim p_z(z)}[log(D(G(z))]$.

The main challenge of GANs is still the training process and is an active area of research as of this writing. As of current, many loss functions have been proposed to improve the aforementioned problems of unstable training and saturated gradients. One widely adopted loss metric, used in many GAN architectures today, is the Wasserstein metric which has been shown to improve both training stability and the optimization process of the model [2].

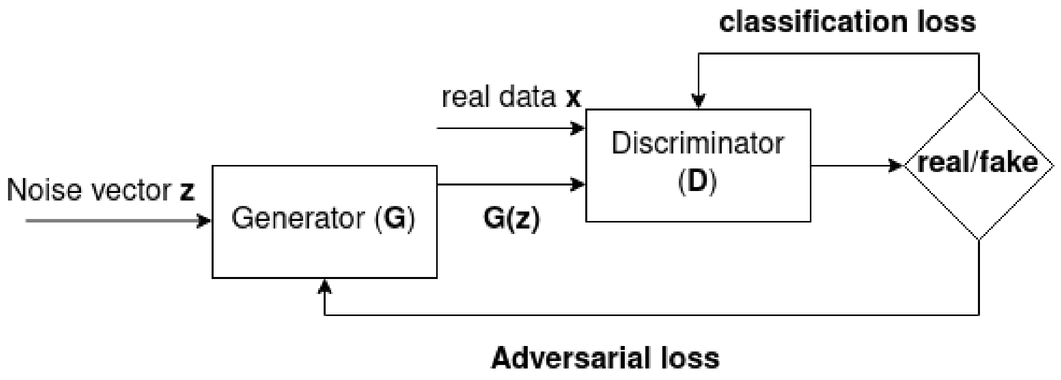


Figure 2.4.1: Diagram over the GAN framework

2.4.1 Wasserstein Distance

The Wasserstein distance, also referred to as the "earth-moving" distance, offers a unique approach to defining the distance between two probability distributions. It seeks to find the optimal transport plan that will shift one distribution P towards another distribution Q. The Wasserstein-1 distance is formally given by the following equation:

$$W(P,Q) = \inf_{\gamma \in \Pi(P,Q)} \mathbb{E}_{(x,y)\sim\gamma} ||x - y|| \tag{2.15}$$

In this equation, $W(P,Q)$ denotes the distance of the optimal transport plan and $\Pi(P,Q)$ represents the joint distribution of P and Q. The symbol inf denotes the greatest lower bound, or in this context, the transport plan that yields the minimum transport cost. Arjovsky et al. (2017) [2] showed that the Wasserstein metric has certain properties in comparison to other metrics used for assessing similarities between probability distributions. Particularly, the gradients of the Wasserstein metric do not saturate or explode during training, which improves the process of model optimization. However, the optimal transport plan expressed in Equation 2.4.1 can be highly intractable due to the infinite possible ways of constructing the transport plan. To overcome this issue, Arjovsky et al. [2] proposed a naive, yet effective solution of applying weight clipping to the Wasserstein metric. GAN architectures have adopted the Wasserstein metric with weight clipping as a loss function, leading to improved training stability and sample quality across multiple architectures, compared to previous loss functions. Gulrajani et al. (2017) [14] further enhanced the Wasserstein loss by introducing gradient penalty instead of weight clipping, which has been demonstrated to further improve the training stability and the quality of generated samples.

2.5 Related Work

A typical domain where the collection of user behavioral data is common is the field of usability testing. The purpose of usability testing is to affirm whether the UI design of a product is satisfactory, ultimately providing a high level of user experience.

Kobayashi et al. [23] constructed a 2-week experiment where 20 elderly participants, ranging in age from 60 to 70, performed four touchscreen operations: tapping, dragging, pinching without panning and pinching with panning. The experiment's goal was to detect possible hurdles that the UI design may pose to the users and record changes in participants' performance during the experimental process. The result was quantitatively presented by calculating the average scores in time taken to complete a specific operation and the accuracy in performing the procedure, e.g., how many taps were required to press a button successfully. A qualitative assessment was done by interviewing the participant regarding possible hurdles when experimenting. The result states that the overall performance of almost all tasks, excluding tapping, was increased during the phase of 2 weeks. The authors also provide suggestions on guidelines regarding design implications that should be considered when designing a UI with respect to the elderly population. A limitation, also stated in the paper, was the absence of younger participants, which removes the possibility of analyzing the difference in performance between age groups.

Tsai et al. [38] constructed an experiment where three sets of groups, children (aged 10-14 years), adults (aged 18-34 years), and elderly (over 60 years) each with 47 participants, performed various UI gesture tasks on three different smartphones, each with a different display size. The examined UI gestures were the dragging task, pinch task, double-touch drag task, multi-long press task, and slide-down task. The study aimed to detect how the display size and the age factor affect the performance of each examined UI gesture. The performance of each UI gesture was recorded using the total time taken to complete the task and the number of attempts before completing the task. The authors used a single-factor ANOVA test to assess the difference between the different groups and found that the age factor significantly influenced all gestures. The elderly were the slowest, and the adults and children showed no statistical significance in performance in completing the gestures. The authors state limitations regarding the generalizability of the gathered results. They stated that the recruitment of participants was limited by time and cost, ultimately leading to participants sharing the same cultural background. Additionally, the authors remarked that all elderly participants were healthy residents without any noticeable health deficiencies; thus, one should be careful to generalize the elderly results with respect to the overall elderly population.

Both [23] and [38] offer insights into usability testing across different user groups and contexts. However, the method used to gather user data has shown to be both costly and time-consuming, which can negatively affect the time for actual research. One approach to reducing the overall cost and time spent on collecting data is to deploy a synthetic data generator whose goal is to replicate the characteristics of the original data. The GAN framework is one proposed solution for synthetic data generation that has been deployed in numerous domains, such as image generation, tabular data generation, and time series generation[20]. Both image generation and tabular data generation do not fit our problem description; usability testing often requires the user to perform a task over a time period, which leaves time series generation as a suitable solution for our problem description. Numerous GAN architectures designed for time series generation have been proposed [27],[6], [41].

Esteban et al. [6] made an initial attempt to adapt the vanilla GAN architecture for handling time series data. They proposed 2 architecture designs, one called RGAN which substitutes the networks within the original GAN with vanilla RNN networks for both the generator and discriminator, and the other named RCGAN which conditions both RNNs within the network using auxiliary information. Both models were trained on a medical dataset that contained records of 200.000 patients across the US, where information about the patients heart rate, respiratory rate, oxygen saturation rate, and mean arterial pressure was measured every five minutes. Furthermore, the dataset were downsampled to one measurement every fifteen minutes by taking the median value of the time window. The authors evaluated the GANs performance based on the maximum mean discrepancy score to compare the similarity between the real and synthetic distributions. Furthermore, the authors proposed the "Train on Synthetic, Test on Real" (TSTR) method to evaluate the GANs performance when used in a classification task. The results indicate that RCGAN produces high fidelity data that produces diversity in samples, meaning that the real and synthetic distributions are statistically similar, while also achieving good performance when trained on the classification problem. The paper do however not consider whether the temporal correlation within the generated samples are followed according to the real dataset. Also, the GAN is trained on a relatively short sequence length (30), and may not produce as good results with longer time series.

Yoon et al. [41] published a new architecture called TimeGAN which aims to resolve prior issues where the only feedback used to learn the underlying distribution was based on the discriminator loss, which does not concern about preserving the temporal dynamics within the synthetic data. TimeGAN introduces 4 kinds of components, the encoder which transforms the time series to a latent space representation, the decoder which transforms the latent space representation back to the time series domain, and the last 2 components being the generator and discriminator, similar to vanilla GAN. The training process involves 2 phases, first phase trains the encoder-decoder networks to help understanding the temporal dynamics within the time series training data. The other phase performs the standard GAN procedure where the generator tries to fool the discriminator with the synthetic samples. The proposed architecture was evaluated on multiple real-world dataset and the GAN performance were compared to other GAN solutions, one being the previously mentioned RCGAN. The results were evaluated using both quantitative and qualitative measurements. For a qualitative assessment, the authors performed t-SNE and PCA analysis to visualize how close the distributions between the real and synthetic samples resembles in a 2-dimensional space. For the quantitative assessment, the authors performed a TSTR test, as previously described, but added an additional task for predictive modelling. The result showed that the TimeGAN architecture consistently outperformed previous GAN architecture across all dataset in both the classification and predictive modelling tasks, indicating a high fidelity in the synthetic data. One limitation of TimeGAN is the lack of long sequence data testing, and also that the architecture does not incorporate attribute information about the time series which can be valuable when individual sequences carry certain characteristics.

Alankar et al. [27] saw the limitations within both [41] and [6], and proposed a new GAN architecture called doppelGANger to further improve the fidelity of long sequence time series generation and also incorporate possible attributes that the time series may carry. The proposed architecture was compared against both TimeGAN and RCGAN using three different, network related, datasets. The authors used the TSTR methodology for a quantitative assessment. For the qualitative measurements, the average auto-correlations alongside various statistical distributions were visualized to further affirm the diversity and quality of the synthetic data. The doppelGANger architecture show great results across all metrics when compared to previous GAN architectures, where the model provides better fidelity on longer sequences, incorporates the auto-correlations present in the real datasets and also show diversity in generated samples. DoppelGANger also outperforms the other GAN architectures when trained on smaller datasets, which is promising for our use-case. In this study, the doppelGANger architecture will be used as the underlying architecture for generating synthetic data. More details about the structure of doppelGANger is presented in section 3.4.2.

Chapter 3

Experimental Design

3.1 Research Question

As discussed in the introduction chapter, collecting user data from various user groups can be challenging in many cases, especially when the groups are hard to access. For instance, the collected data may be used to improve the quality of an existing product. The limited data which reflects the user groups may not be enough for a proper analysis of various trends and patterns in the dataset. This can negatively affect the analysis of the dataset as the product may be more adapted to the commonly occurring group. The goal of this study is thus to examine to which degree GANs can be used to generate synthetic user behaviour of various user groups. To tackle the goal of this study, we introduce two set research questions, **RQ1** and **RQ2**, which combined aims to answer whether one could generate sufficiently good synthetic data of various user groups when interacting with a mobile user interface, using GANs. The set research questions are the following:

- **RQ1**: How well does synthetic data resemble UI interaction data for different user groups?
- **RQ2**: How well does the synthetic UI interaction perform compared to real user interaction data?

The upcoming sections will describe the various design decisions and techniques used to help answering the set research questions.

3.2 Experimental Overview

The experiment comprises several stages designed to address the research questions mentioned earlier. Before starting the experiment, we have to define what type of action the user should perform, that is, the UI interaction we want to capture and what kinds of user groups the study mainly focuses on. The UI interaction is a simple drag-and-drop operation where the user drags a target widget to a designated destination

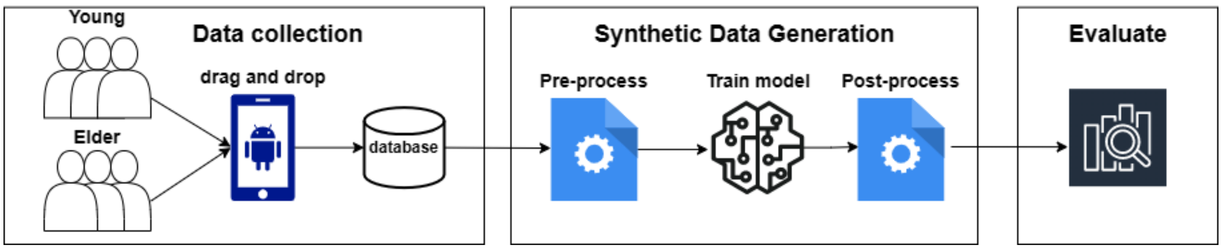


Figure 3.2.1: Overview of experimental stages during project

on the screen. The user groups on which the data collection is based are the young and elderly populations.

As illustrated in Figure 3.2.1, the initial stage involves collecting the drag-and-drop sequences via a custom-developed Android application. Before the commencement of each experiment, participants complete a questionnaire about their age, dominant hand, and the finger they most commonly use when operating a smartphone. Upon finishing the questionnaire, participants start with the drag-and-drop sequences. A single experiment consists of 15 consecutive runs, where each run is a single drag-and-drop sequence.

The gathered data is subsequently consolidated into a single dataset, encompassing all experiment runs. This dataset is further divided into two sub-groups: the elderly and young populations. These two datasets are then independently pre-processed for training the generative model. Following training, two SDGs will be available, each generating synthetic drag-and-drop sequences for the specified user group.

The models will be employed to generate datasets, which we will subsequently combine and assess using quantitative and qualitative measurements. The analysis aims to ascertain the quality of the synthetic data and its performance relative to the real data, ultimately addressing the previously mentioned research questions.

3.3 Data Collection

The initial step in all machine learning projects involves obtaining the relevant data, which can be used to fit an arbitrary model in order to achieve a specific goal. In our case, the data should represent user behaviour when interacting with a phone screen.

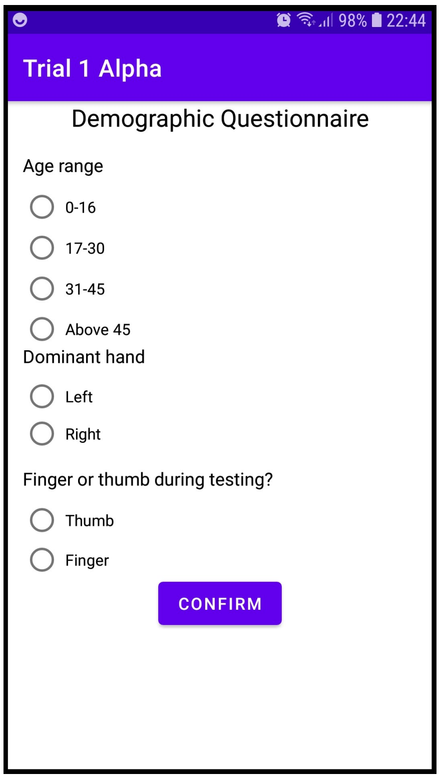
This study primarily focuses on the drag-and-drop operation, during which the user moves an arbitrary square to a predetermined destination. Other typical user interactions could be tested, such as pressing buttons of various sizes or pinching widgets within a specific range. We believe that if one successfully trains a model using the drag-and-drop technique, the suitability of using GANs in other settings should not exhibit significant differences. Additionally, we divide the main dataset into two separate datasets, each representing a specific user group. The motivation for splitting the dataset between the user groups is that each dataset will be pre-processed individually and used for training a GAN according to the pre-processed dataset. In this study, we split the dataset based on participants' age.

3.3.1 User Task

The data used in this project was collected using a custom-developed Android application which was designed to capture statistical information about the participant, as well as their interaction when performing the drag-and-drop operation. The application starts by providing the user with a questionnaire, depicted in figure 3.3.1, which is mandatory to complete before starting the actual tests.

After completing the questionnaire, the user is prompted with two squares, one labeled DRAG and the other DROP. The task is to move the DRAG square to the DROP location as depicted in figure 3.3.2. A timer starts only when the user starts moving the DRAG square and stops only when reaching the DROP. When the user successfully drops the square in the designated position, the positions of the squares are randomly moved across the UI and a new run of the experiment is started. The user performs a total of 15 drag-and-drop sequences.

When the experiment is completed, a sequence of coordinates ordered over time, reflecting the drag and drop, alongside the questionnaire are stored in separate tables inside a database. Both tables are connected using a unique id that each participant is assigned at the start of an experiment.


98% 22:44

Trial 1 Alpha

Demographic Questionnaire

Age range

○ 0-16

○ 17-30

○ 31-45

○ Above 45

Dominant hand

○ Left

○ Right

Finger or thumb during testing?

○ Thumb

○ Finger

CONFIRM

Figure 3.3.1: The questionnaire which the user has to complete before starting the experiment

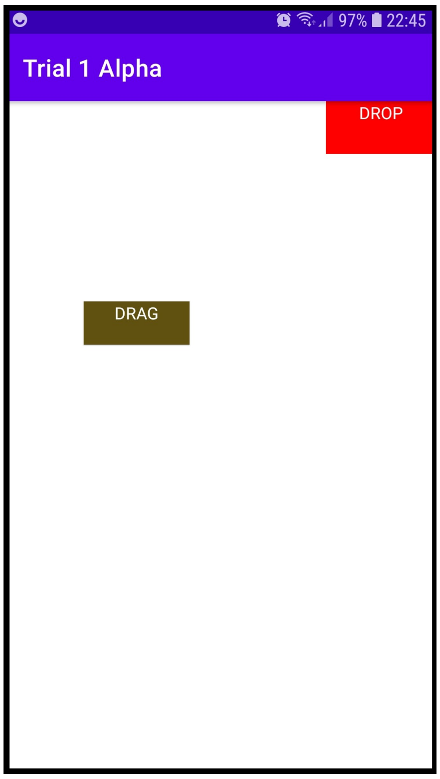


Figure 3.3.2: The user has to move the square to the designated drop location

3.3.2 Collected Data

Drag-and-drop

Each drag-and-drop operation can be viewed as a continuous alteration in the x and y coordinates over a time period until the destination is reached, as illustrated in Figure 3.3.4. The rate at which the coordinates are recorded is set to 10 ms, signifying that a total of 100 data points are collected per second. We believe this rate is sufficient to capture the majority of user interactions. As the data points are gathered over a time period, the comprehensive dataset will comprise multiple time series, with each sample representing a single drag-and-drop sequence. Even if the user drops the square outside the designated drop location, the coordinates are still captured each time frame. Important to note is that each sample may have varying lengths due to the randomly placed starting and destination positions of the drag-and-drop and the participant's individual ability to complete the task in a given time frame.

A single drag-and-drop sequence is depicted in figure 3.3.3 where the starting $(x_{start}, y_{start}) = (800, 0)$ and the destination $(x_{dest}, y_{dest}) = (400, 890)$ over 110 time-steps where each time-step is 10ms. The blue and orange line shows the change in both x and y coordinate over the given time frame.

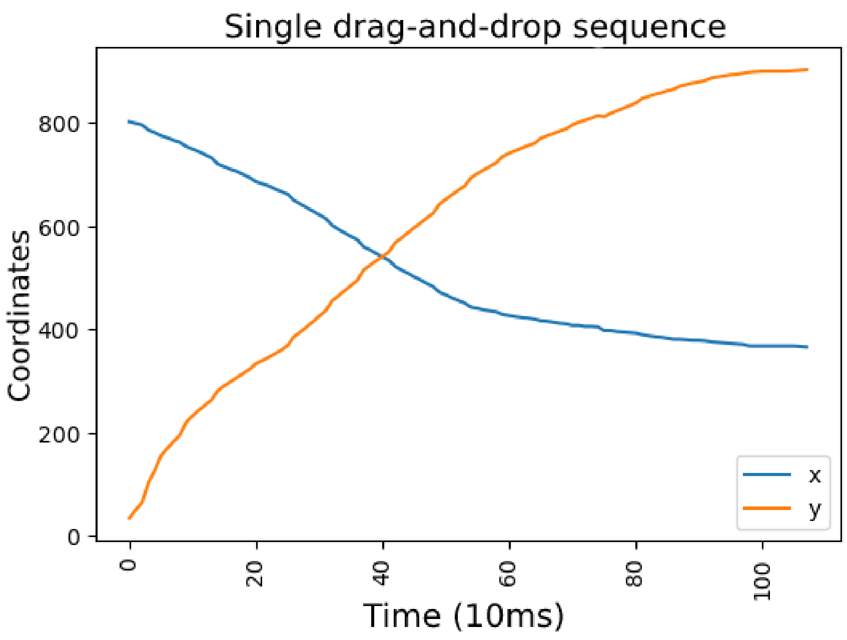


Figure 3.3.3: A single drag-and-drop sequence from start to finish

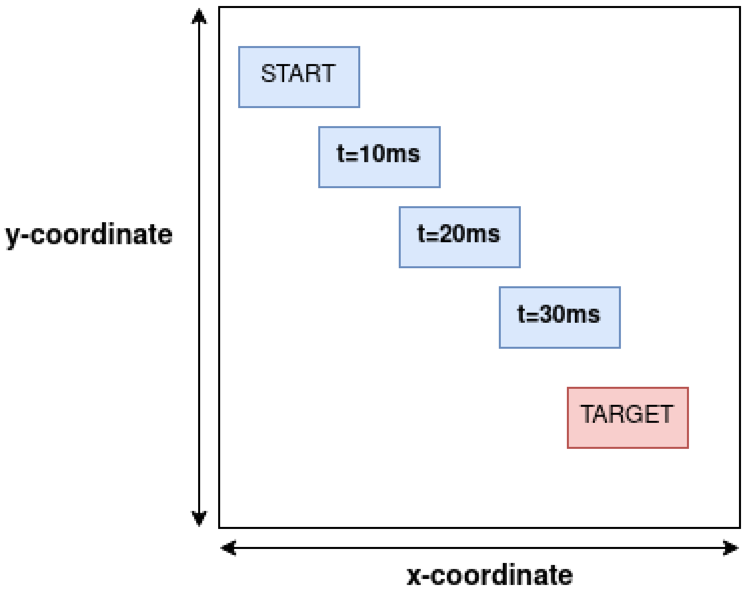


Figure 3.3.4: Example of drag-and-drop sequence where the starting square is displaced to the target destination. The coordinates are captured at each time step

User Statistic

In addition to the aforementioned time-series data, the dataset also encompasses information that remains constant at each time step. In this study, we collect data pertaining to participants' age, dominant hand, and finger used during testing. The dominant hand and finger utilized are of a boolean nature, signifying either the left or right hand, and finger or thumb, respectively.

Conversely, the age variable is divided into four groups, ranging from 0-16, 17-30, 31-45, and above 45. This metadata is employed to examine biases and trends within the dataset. Furthermore, we utilize the metadata to differentiate between various user groups, in this instance, segregating between the elderly and younger populations.

3.4 Synthetic Data Generation

3.4.1 Pre-processing

The pre-processing of the dataset is a crucial step in ensuring that our model achieves optimal performance during its training phase. The initial step in our pre-processing involves removing outliers from the raw dataset. Subsequently, we must pad the time series to equal lengths for model training. Lastly, the time-series coordinates are normalized prior to the training process.

Remove Outliers

A common occurrence when collecting data from arbitrary sources is the presence of outliers in the dataset. An outlier can significantly impact the quality of our dataset when attempting to fit it into a machine-learning model. This is particularly true in our case when handling time-series data, as we must pad all time series according to the longest sequence. For example, if the average number of time steps across all time series is 150 and an outlier within the dataset has 400 time steps, the majority of time series must be padded with 250 empty points, which can negatively affect the training process later on.

The approach used to remove all outliers from the dataset involved excluding all time series with a length larger than the fourth quartile, $Q4$. To obtain the fourth quartile, we first had to gather the individual lengths of each time series and calculate the quartiles $Q1$ and $Q3$, where $Q1$ is the highest value in the 25% percentile range, and $Q3$ is the highest value within the 75% percentile. By determining these points, we can calculate the fourth quartile using the following equation: $Q4 = Q3 + 1.5(Q3 - Q1)$.

Padding Time-series

As previously mentioned, each drag-and-drop sequence may vary in length which gives us an inconsistency in our feature representation. All machine learning models are essentially a function $f : X \to Y$, where $X \in \mathbb{R}^n$ and $Y \in \mathbb{R}^m$. The machine learning model thus requires a fixed length input which we can achieve if all our time series are of the same dimension. One solution to this problem is to pad all time series with an arbitrary value til reaching the length of the longest occurring sequence, excluding the outliers. For instance, imagine if our dataset X consists of 3 samples with the following elements respectively:

$$X = [[15, 23], [6, 103, 5], [1, 3, 10, 15, 54]]$$

The longest sequence inside X is the last sample $X_3 \in \mathbb{R}^5$ while $X_1 \in \mathbb{R}^2$ and $X_2 \in \mathbb{R}^3$. Applying padding to this dataset yields the following new dataset:

$$X_{padded} = [[15, 23, 0, 0, 0], [6, 103, 5, 0, 0], [1, 3, 10, 15, 54]]$$

Where X_1 and X_2 are padded with 0. It is essential to consider that the value utilized for padding the dataset should not conflict with existing values within the dataset, as it is necessary to distinguish the starting point of padding within a given sequence to retain the prior knowledge of our original dataset.

In our dataset, we introduce a new feature alongside the existing x and y coordinates called pad which has either 1 indicating not padded and 0 indicating padded. This new feature's sole purpose is to indicate whether the current (x, y) pair is padded or not in a particular time series.

When applying padding to the young and elderly datasets, we must consider the longest time series excluding outliers. The longest time-series sample within the young population dataset is 265, while the longest for the elderly population dataset is 410. Consequently, all samples within each dataset are padded to the lengths of 265 and 410, respectively.

Normalization

Normalization is a widely used technique for scaling a dataset to values within a bounded interval. The benefits of a normalized dataset include consistent values across the dataset, which stabilizes gradient-based learning and facilitates faster model convergence [36].

In our case, the x and y coordinates within the dataset are constrained by the dimensions of the phone used during the experiments. The x and y values can significantly vary among the time series within the dataset, primarily due to the random displacement of each run. To ensure consistency in our dataset, we normalize the coordinates such that $x, y \in [-1, 1]$ by employing the following formula:

$$x_{norm} = 2\frac{x - x_{min}}{x_{max} - x_{min}} - 1 \tag{3.1}$$

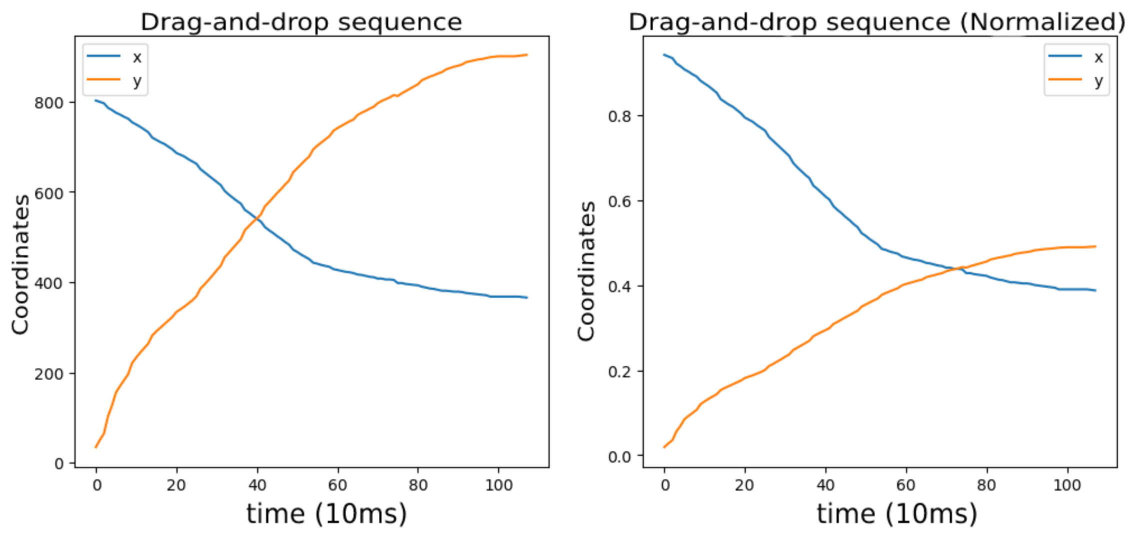


Figure 3.4.1: Left shows the non-normalized sample within the original dataset and the right shows the same sample but normalized.

It is crucial to emphasize that we apply 3.1 to each x and y vector individually within the dataset. This process is demonstrated in Figure 3.4.1, where the original dataset, comprising all time series, has been normalized in accordance with 3.1.

3.4.2 GAN Architecture

This project uses the doppelGANger architecture, depicted in figure 3.4.2, as the underlying network for generating the synthetic data. The motivation behind using doppelGANger instead of other existing architectures [6, 41] was mainly due to worse fidelity when the GANs were trained on long sequence data [27].

Furthermore, we introduce two distinct GANs, one for the younger population and another for the elderly population. Both GANs share the same underlying architecture (DoppelGANger) but differ in (1) the input dimensions and (2) the configurations of each component within the architecture. The choice of settings was based on manual tuning of the individual components and observing their impact on model performance, as well as guidelines from the original author of DoppelGANger [27].

The architecture uses five distinct networks, the metadata generator, the time series generator, the min/max generator, the auxiliary discriminator, and the primary discriminator.

In this project, we exclude the use of the metadata generator. Because we introduce 2 separate GANs, one for the elderly and younger populations, there is no need to generate attributes that correlate each generated time series with the corresponding user group.

For the min/max generator, we use a dense MLP with ReLU activation at each layer and a single output value. The input to the network is a noise vector drawn from a Gaussian distribution and the output is the metadata that we want to generate. In our case, the metadata is the $(min \pm max)/2$ value we try to learn for each sample within the dataset. This metadata value is then used as a conditional value to the time series generator input, meaning that we generate the drag-and-drop sequence, given the metadata. The metadata is used as an internal, per-sample normalization to preserve the fidelity of the generated samples by learning the ranges of each individual sample within the dataset. This has been shown to improve the quality of the data but also mitigate the consequence of a mode-collapse [27]. Mode collapse is the consequence when the GAN produces similar to nearly identical samples for different input noise vectors [11].

The time series generator uses an RNN network with LSTM units to generate the drag-and-drop sequences. The input is the previously generated $(min \pm max)/2$ value alongside the noise vector drawn from a Gaussian distribution. Depending on the settings used, one could generate multiple sequences per RNN unit, which is also referred to as a batch generation. The authors of [27] claim empirical evidence that a batch generation of 5 and higher improves the model's ability to capture temporal correlations within each sample for longer sequences. Different batch generation sizes are tested and evaluated based on the results.

The auxiliary discriminator is a dense MLP using the ReLU activation at each layer. The model takes the implicitly generated metadata, that is, the $(min \pm max)/2$, and compares it with the real sample $(min{\pm}max)/2$ value. The auxiliary discriminator aims to judge whether the min/max generator has learned the underlying $(min \pm max)/2$ distribution of the actual dataset. To compare the values, we use the Wasserstein-1 metric with gradient penalty as our loss function; see section 2.4.1. The discriminator, similar to the auxiliary discriminator, also uses a dense ReLU MLP but instead takes the output of the time series generator and compares it with the real drag-and-drop sequences. The loss value is the difference between the synthetic sample distribution and the real sample distribution, which is also calculated using the Wasserstein-1 metric with gradient penalty.

The auxiliary discriminator and the discriminator's loss values are combined into a single value, the total loss of the GAN network. The loss value of the auxiliary discriminator can be scaled to increase/decrease its overall effect when updating the complete network. When updating the model's parameters, we use the Adam optimizer, which is an efficient extension of the stochastic gradient descent algorithm that has been widely adopted as a good baseline optimizer [22].

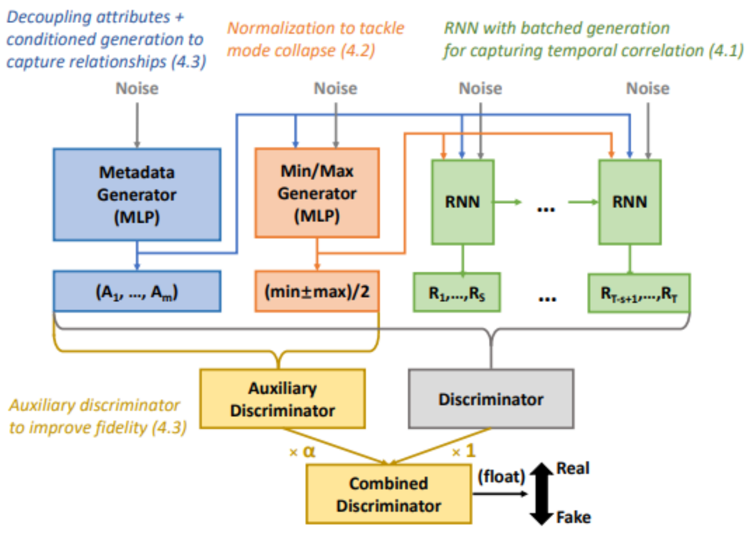

Decoupling attributes + conditioned generation to capture relationships (4.3)
Normalization to tackle mode collapse (4.2)
RNN with batched generation for capturing temporal correlation (4.1)
Noise
Noise
Noise
Noise
Metadata Generator (MLP)
Min/Max Generator (MLP)
RNN
...
RNN
(A1, ..., Am)
(min±max)/2
R1,...,RS
...
RT-S+1,...,RT
Auxiliary discriminator to improve fidelity (4.3)
Auxiliary Discriminator
Discriminator
× α
× 1
Combined Discriminator
(float)
Real
Fake

Figure 3.4.2: Overview of the doppelGANger architecture [27]

3.4.3 GAN Training

When training both GANs, we need to specify the number of epochs and the batch size used during training. The batch size represents the number of samples employed to update the model's internal weights during one training iteration. For instance, if we use a batch size of 1 and the dataset contains 100 samples, the model's internal weights are updated 100 times per epoch. In both GANs, we utilize a batch size that includes all samples of the dataset, primarily due to the small datasets used during the experiments. Employing a larger batch size also mitigates the otherwise fluctuating loss values that are common when using smaller batch sizes. The number of epochs is set to an arbitrary number and then incrementally increased or decreased based on the model's performance. The models' performance is evaluated using various techniques, which are discussed later in this chapter.

3.4.4 Post-Processing

After training the GANs, we can now prompt both the attribute and time series generators, using a random input vector, which allows us to produce an arbitrary amount of synthetic drag-and-drop sequences. The raw generated data does however require additional processing before we can evaluate its quality. First, we have to re-normalize the coordinates back to the original ranges. This is easily done by using the normalization formula in 3.1 and solve for x which yields:

$$x = (x_{norm} + 1)\frac{(x_{max} - x_{min})}{2} + x_{min} \tag{3.2}$$

Note that we have to store the minimum and maximum occurring values in the dataset during the pre-processing step.

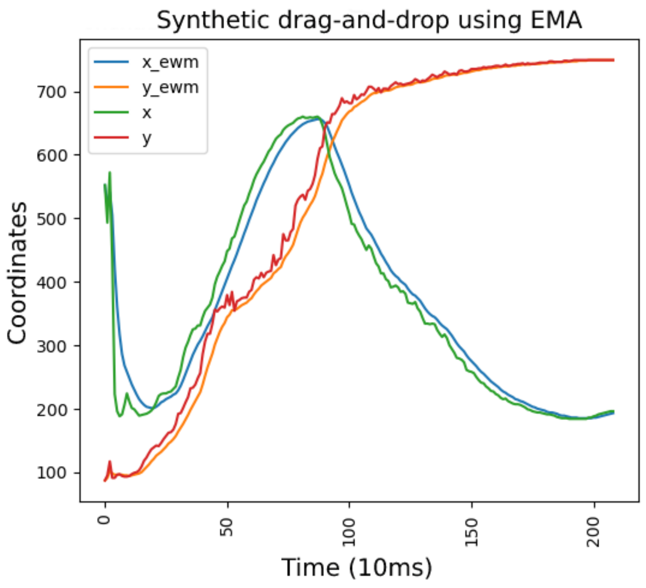


Figure 3.4.3: EMA applied over the generated coordinates

Alongside reverting the values back to the original ranges, we also have to remove any potential padding that the generated data may include. Recall that in section 3.4.1 we introduce a new feature called pad which the GAN is also trained to learn. To remove padding, we remove all entries inside a single sample which has $pad = 0$.

The last step in our post-processing is to apply a moving average to the synthetically generated coordinate sequences. The motivation behind using a moving average to the data was to remove any noise which the generators may produce. For our purposes, we use a exponential moving average (EMA) which is commonly used when we want to weigh more emphasis on the most recent data points. To calculate EMA on a given sample, the following formula is used:

$$y_t = (1 - \alpha)y_{t-1} + \alpha x_t \tag{3.3}$$

where y_t is the calculated EMA at time t, α is the smoothing factor which specify how much emphasize the most recent observation should have. α must be a value between $0 < \alpha \leq 1$ and x_t is the data point at time t. Figure 3.4.3 shows the result when applying EMA over a synthetically generated drag-and-drop sequence where α is set to 0.2.

3.5 Evaluating the Synthetic Data

Evaluating the GANs performance has shown to be very hard compared to other domains in machine learning [6, 13, 27, 41]. Normally in many domains, we can evaluate our model's performance based on the calculated loss values at each epoch. If the loss converges towards a small value, we may assume that the model has properly learnt the underlying distribution. In the case of GANs, the loss value is calculated based on the discriminator's performance in distinguishing real from fake samples. The generator uses the discriminator's loss value to adjust its internal weights, hence the generator uses an implicit loss that is not objectively based on the quality of the fake samples. Thus to ensure that the generator inside the GAN provides good samples, we have to combine both quantitative and qualitative measurements.

In this study, the aim of the quantitative measurements is to assess how well the synthetic samples perform compared to the real dataset. To achieve this, we employ an RNN model for a simple classification task.

The qualitative measurements are intended to analyze whether the generated output follows the same trends and patterns as the original data. This is accomplished by using various visualization plots to depict the statistical properties of the overall datasets. Additionally, to determine whether the synthetic data exhibits diversity in the generated samples, we select random samples within the synthetic dataset and examine how closely they align with the samples in the real dataset.

The upcoming sections will delve into more specifics on what methods are used to answer the research questions **RQ1** and **RQ2**.

3.5.1 RQ1

To answer **RQ1**, we rely on various techniques to detect whether the generative model has successfully learned the underlying distribution of the original dataset.

Because our dataset is of time series format, we have to consider the temporal correlations within the samples when analyzing the datasets. One approach which we use to analyze whether the synthetic dataset shares statistical similarities with the real dataset was to calculate the average delta distance $\triangle d$ from the current drag position (x_i, y_i) to the designated drop location (x_{drop}, y_{drop}) at each timestamp t_i. Another aspect to consider is how well the synthetic data follows the per sample length distribution. This is simply done by removing the padding of each sample within the synthetic dataset and compare the resulting sample lengths of both datasets.

Lastly, to ascertain whether the synthetic dataset demonstrates diversity in its samples, we calculate the k-nearest neighbors of a randomly selected synthetic sample and compare it to the samples in the real dataset. We use a k-value of 3, meaning that we extract three samples in the real dataset. Dynamic time warping (DTW) is used to find the nearest neighbors withing the datasets. DTW is an algorithm which is commonly used to measure the similarity between two time series with varying lengths, which is common in our datasets. In its essence, DTW seeks to align the compared time-series such that their euclidean distance minimizes [37]. The samples that results in the least distance relative to the synthetic sample is considered its nearest neighbor.

3.5.2 RQ2

The technique used to test the quality of the synthetic data is based on the train on synthetic, test on real (TSTR) methodology[7]. The method uses an external machine-learning model trained on synthetically generated data and evaluated based on the real data. This approach aims to determine whether the synthetic data can be used to either replace or augment the existing dataset based on the scores achieved when evaluating the model. In this experiment, we train an RNN model for a classification task. The classification task is a simple binary classification where we try to train the model to distinguish between the younger and elderly samples. Figure 3.5.1 depicts the workflow on how TSTR is designed for our classification experiment. First, we have to generate the samples for the respective user group using the already trained GANs using a noise vector **z**. The samples are then merged into one dataset, which is pre-processed similarly to how the datasets were pre-processed when training the GANs, except that we also add a field for labeling the samples as either elderly or young. We then use the synthetic dataset to train the model, which is chosen as a simple RNN with LSTM units. The trained model is then tested based on the real dataset and the performance is evaluated based on its recall, precision and F1 score. The recall is defined with the following formula:

$$Recall = \frac{TP}{TP + FN} \tag{3.4}$$

Where TP stands for the true positives, FP for the false positives and FN for the false negatives. The precision is based on the following formula:

$$Precision = \frac{TP}{TP + FP} \tag{3.5}$$

and the F1 score, which is the harmonic mean of the precision and recall scores, is

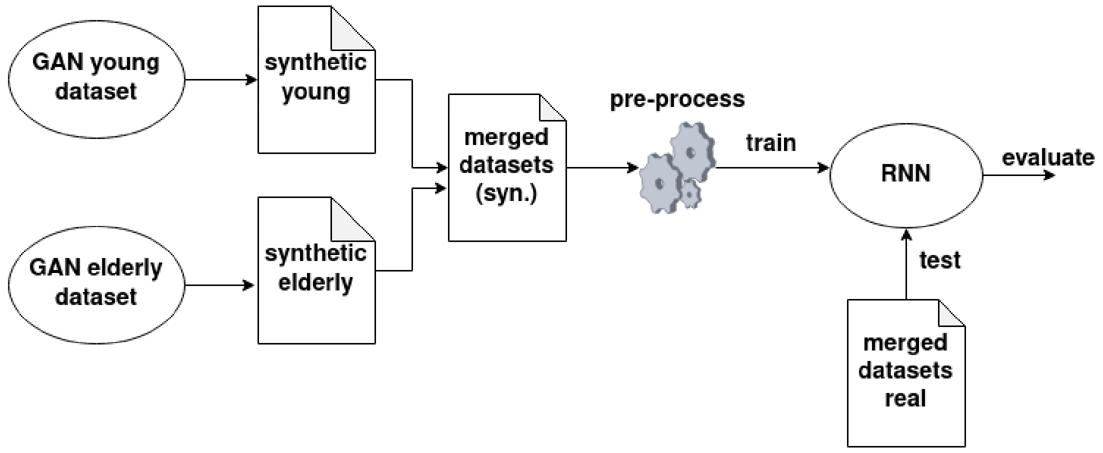


Figure 3.5.1: High-level overview of TSTR used in our experiment

calculated using the formula:

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{3.6}$$

Alongside training a RNN on synthetic data, we also train the same RNN using the real dataset which provides a baseline for comparison. If the metric scores are similar across both the synthetically trained model and the real, we may assume that the quality of the synthetic data is similar to that of the real data.

Chapter 4

Implementation

4.1 Overview of Implementation

The implementation consists of 2 separate parts, the mobile application and the experimentation platform. The mobile application was developed using Android Studio and the code was written using Java. The experimental playground was deployed using docker. Docker is a virtualization service that builds sandboxed containers that run on top of the docker engine [5]. Docker was used to providing a platform-independent playground that enables fast and shareable results, ultimately mitigating the problem of missing dependencies when one tries to replicate the experimental results. Within the container instance, we use JupyterLab to construct the experiment regarding synthetic data generation. As depicted in figure 4.1.1, a total of 5 notebooks were used to construct the experiment. The *database_fetcher* is responsible for retrieving the data from the database, alongside converting the raw data to a Comma-separated Values (CSV) file. The CSV-file is then processed inside of *user_statistic*, where we remove outliers and divide the CSV-file into two separate files, one representing the younger and elderly population. The *GAN_elderly* and *GAN_young* notebooks are responsible for the pre-processing and training of the GANs. The *evaluate* notebook contains the various qualitative and quantitative evaluation techniques we use to determine whether the GAN produces satisfactory results. All notebooks were written using Python 3.8.

4.2 Data Collection

4.2.1 Mobile Application

To gather data, a mobile application had to be developed to (1) gather the drag-and-drop sequences (2) gather user information such as participants age and (3) store the data in a database.

As previously mentioned in section 3, we wish to capture the drag-and-drop operation

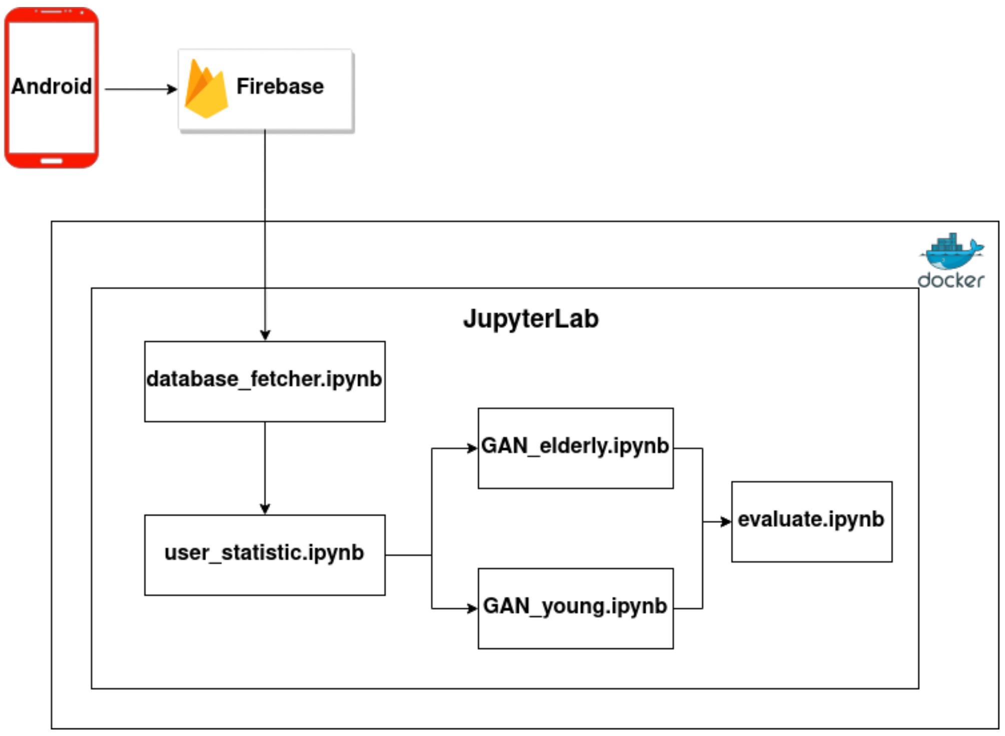


Figure 4.1.1: Overview of the implementation

of various users, which we later use to train the GAN model. Figure 4.2.1 display the logical order in how the we capture the drag-and-drop sequences for each participant. When the user finishes the questionnaire, a new page will appear with 2 squares, a draggable square which we label ”DRAG” and the target square which we label ”DROP”. To capture the dragging sequence, we use Androids Drag and Drop API, which provides useful event and callback functionality that we can use to detect various user actions but also gather information about the given interaction, such as the current coordinate position on the screen. The main events we listen to is (1) the initial interaction with the square, and (2) when the user drops the square, and (3) when the user drags the square.

When the user starts dragging the square, an event is fired which captures the (x, y) coordinate at irregular time frames. To ensure that we get evenly spaced capture points, we employ a external timer that is called every 10 ms. When the timer is fired, we store the currently captured coordinates in a list. When the user drops the square outside the designated location, we store the last recorded finger position each 10 ms. When the square is dropped in the target section, we check whether the number of iterations are complete, if not, randomly position the squares and start a new iteration. If the number of iterations are complete, we store all recorded drag-and-drop sequences in a remote database. The number of iteration was set to 15, meaning that a single participant provides 15 unique time series samples.

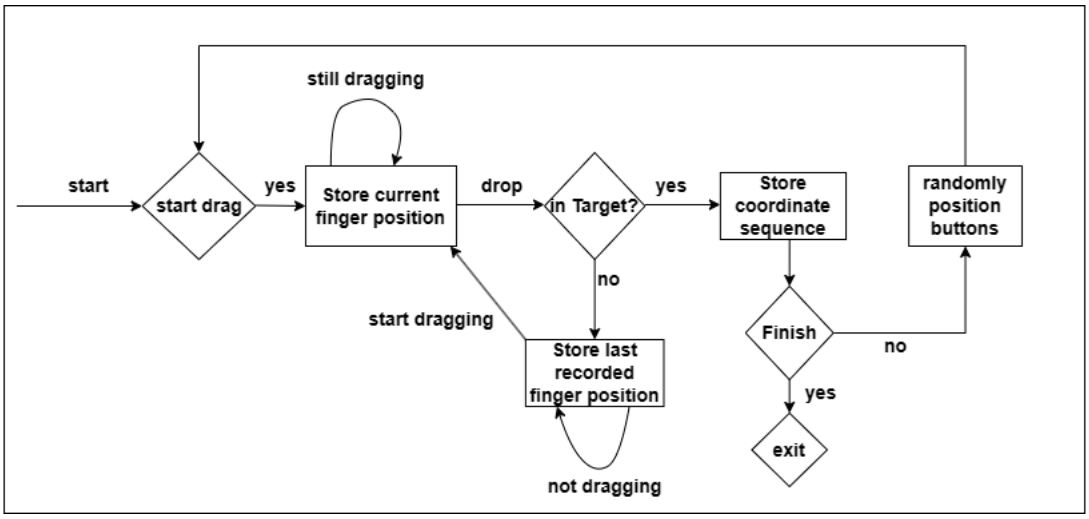
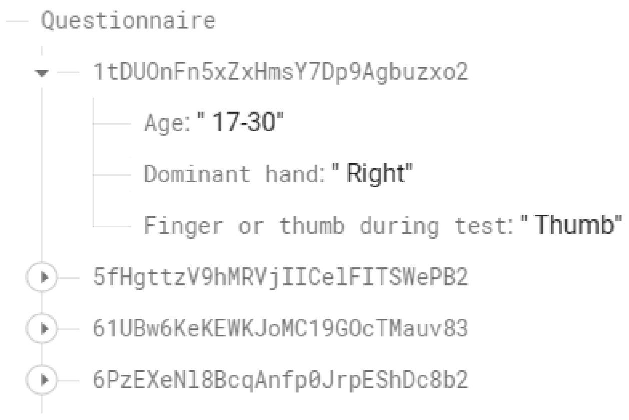

still dragging
start
start drag
yes
Store current finger position
drop
in Target?
yes
Store coordinate sequence
randomly position buttons
no
start dragging
Store last recorded finger position
Finish
no
yes
exit
not dragging

Figure 4.2.1: Workflow of drag-and-drop experiment

4.2.2 Data Storage

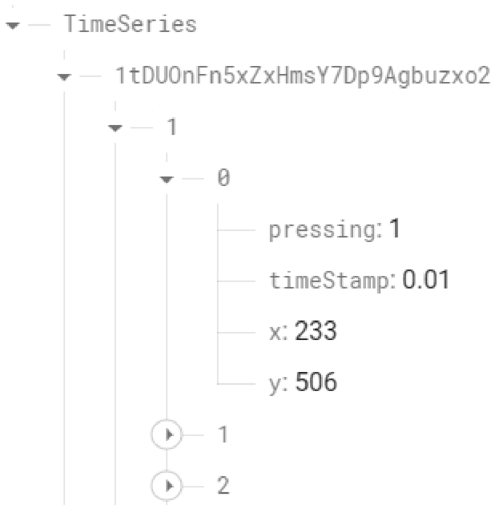
We use Google's database service Firebase to store the time series data. Firebase provides an easy-to-use API to store the time series and questionnaire data efficiently. Alongside the easy-to-use API, the Firebase service is also free to use until reaching a certain data storage threshold. Firebase stores values in JSON format, and each entry is referred as a node. Nodes inside the database can have child nodes, enabling easy querying when extracting the data. We introduce two root nodes to store our data: the time series data and the questionnaire data.

Figure 4.2.2 shows how we store individual entries of each participants and their corresponding information. Each participant, prior to starting the experiment, is assigned a Universally Unique IDentifier (UUID). UUID is a 128-bit number that guarantees uniqueness across both space and time [25]. We use the UUID to separate each entry within the database. The UUID node contains 3 child nodes that store the information about the participant's age, dominant hand, and whether a finger or thumb was used during the experiment. For the time series storage, depicted in figure 4.2.3, we also use the UUID to bundle all individual drag-and-drop sequences. Each UUID has fifteen child nodes that correspond to a single drag-and-drop sequence. As a reminder, we store fifteen entries because that's the total number of iterations each participant performs. Within each iteration, all data points, captured at a 10ms interval, are stored. Each data point contains information about the current (x, y) value, whether the user is dragging or not, and the timestamp in which the coordinates were captured in. Note that each entry may contain various lengths, meaning that e.g. iteration 1 may contain 150 data points while iteration 2 contains 200 data points.



```
Questionnaire
  1tDUOnFn5xZxHmsY7Dp9Agbuzxo2
    Age: " 17-30"
    Dominant hand: " Right"
    Finger or thumb during test: " Thumb"
  5fHgttzV9hMRVjIICelFITSWePB2
  61UBw6KeKEWKJoMC19GOcTMauv83
  6PzEXeNl8BcqAnfp0JrpEShDc8b2
```

Figure 4.2.2: Figure shows how the questionnaire information is stored in Firebase



```
TimeSeries
  1tDUOnFn5xZxHmsY7Dp9Agbuzxo2
    1
      0
        pressing: 1
        timeStamp: 0.01
        x: 233
        y: 506
      1
      2
```

Figure 4.2.3: Figure shows how the drag and drop information is stored in Firebase

4.3 Synthetic Data Generation

4.3.1 Dataset

The data stored in the database is retrieved and bundled into two CSV files (Comma-Separated Values), one for the younger population and the other for the elderly. Both files are identical in structure and contain the time series entries and the questionnaire information. To access and further process the data, we utilize Pandas, a popular library widely used in data science due to its comprehensive data manipulation and analysis toolkit [30]. When reading the CSV files using pandas, we attain DataFrames, which is essentially a 2D table that contains columns, the labels of our dataset, and rows, which are the entries within that table. In figure 4.3.1, we see the DataFrame that represents the collected data of the elderly population. We use the ”Long” format to represent each sample, meaning that each row corresponds to a single time-point, and all samples are stacked sequentially on the vertical axis.

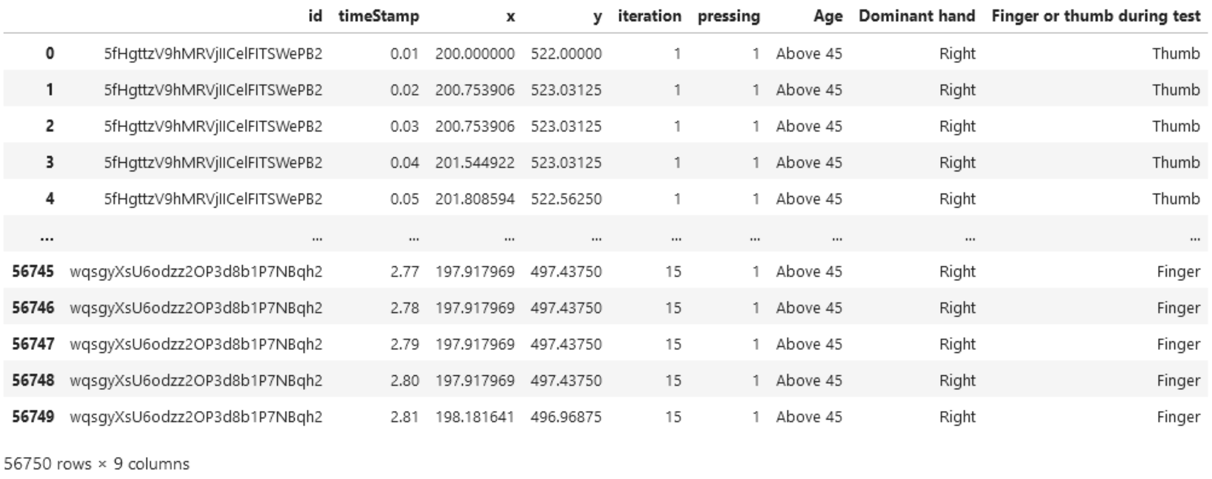
4.3.2 Pre-Processing

Remove Outliers

To remove outliers, we first have to count and store each sample length within the DataFrame. By using the id column alongside the iteration number, as depicted in figure 4.3.1, we can query each sample and store its corresponding length. To calculate the quantiles of the list, we use numpys *quantile* function to attain the forth quantile, $Q4$. We use $Q4$ to drop all samples within the DataFrame that has a length > $Q4$.

Padding Dataset

After removing the outliers in the original dataset, we need to assure that all samples are of equal length. First, we attain the longest occurring sample within the dataset. After obtaining the longest length, we add a new column to our existing DataFrame



	id	timeStamp	x	y	iteration	pressing	Age	Dominant hand	Finger or thumb during test
0	5fHgttzV9hMRVjIICelFITSWePB2	0.01	200.000000	522.00000	1	1	Above 45	Right	Thumb
1	5fHgttzV9hMRVjIICelFITSWePB2	0.02	200.753906	523.03125	1	1	Above 45	Right	Thumb
2	5fHgttzV9hMRVjIICelFITSWePB2	0.03	200.753906	523.03125	1	1	Above 45	Right	Thumb
3	5fHgttzV9hMRVjIICelFITSWePB2	0.04	201.544922	523.03125	1	1	Above 45	Right	Thumb
4	5fHgttzV9hMRVjIICelFITSWePB2	0.05	201.808594	522.56250	1	1	Above 45	Right	Thumb
...	...	...	...	...	...	...	...	...	...
56745	wqsgyXsU6odzz2OP3d8b1P7NBqh2	2.77	197.917969	497.43750	15	1	Above 45	Right	Finger
56746	wqsgyXsU6odzz2OP3d8b1P7NBqh2	2.78	197.917969	497.43750	15	1	Above 45	Right	Finger
56747	wqsgyXsU6odzz2OP3d8b1P7NBqh2	2.79	197.917969	497.43750	15	1	Above 45	Right	Finger
56748	wqsgyXsU6odzz2OP3d8b1P7NBqh2	2.80	197.917969	497.43750	15	1	Above 45	Right	Finger
56749	wqsgyXsU6odzz2OP3d8b1P7NBqh2	2.81	198.181641	496.96875	15	1	Above 45	Right	Finger

56750 rows × 9 columns

Figure 4.3.1: DataFrame of the elderly population dataset

called *gen* and initialize all rows within this column with 1. Each row with $gen = 1$ indicates a unpadded row. When performing the padding, we extract each unique sample within our DataFrame using the id and iteration columns. After extracting the sample, we iterate $max\_sequence\_length - len(sample)$ times and in each iteration we add the last row of the original sample, which is the occurrence of the user dropping the square in the target. Additionally, we replace the gen value with 0 to indicate that the newly added row is used for padding.

Feature representation

To proceed training our GANs, we first have to convert the current padded DataFrame to a vector which the model can interpret. First, we retrieve the relevant columns of our DataFrame, namely the x, y and gen columns. This leaves us with a $N \times (x, y, gen)$ table where N is the same number of rows as in the padded DataFrame. The table is then reshaped to a three-dimensional vector with the following structure: $(\#of samples, max\_seq\_length, features)$ using numpys *reshape* function. Now, the first column represents a unique sample, the second column the sequence values of the sample and the third column the x, y, gen features. For example, $(n, 100, 0)$ attains the first 100 x-coordinates of the n:th sample within the vector.

4.3.3 GAN Model

The GAN model is implemented using gretel.ai open-source repository [9]. The repository provides the class *DGAN*, which implements all models within the doppelGANger architecture. The DGAN-class expects a configuration file in its constructor which is used to adjust the DGAN-model during its initialization. The configuration class is called *DGANConfig* which contains variables to adjust the parameters within a certain model inside DGAN, e.g. the time series generator. Alongside adjusting the structure of individual models inside the architecture, we can also specify the number of epochs used for training and the batch size, which specifies the number of samples the model uses during a single epoch.

Alongside the baseline model, the repository also provides functionalities for the normalization of the feature vectors prior to training the model, which is used to scale the features in ranges between $[-1, 1]$.

Training

After initializing the DGAN-class, using the specified settings, we get a model ready for training. The DGAN-class provides a function called *train_numpy* which is used to train the DGAN-model. The function expects a feature vector alongside a list that specifies the individual feature type. The provided feature vector is the three-dimensional vector representation of our padded dataset. The feature types are used to determine whether the features are continuous or discrete. As previously mentioned, we have 3 features (x, y, gen), the (x, y) coordinates are set to continuous while gen is set to discrete. The feature types will ensure that the output vector, produced by the generator, follows the same type convention as our original feature vector.

Generate Samples

After training the model, synthetic samples can now be produced by providing a random noise vector to the time series generator. This process is abstracted by the *DGAN* class, which provides a function called *generate_numpy* that takes an integer n, as an argument, which results in n generated samples. The generated samples have the same dimension as the three-dimensional vector which we use to train the model. Additionally, the function also handles the re-normalization of the samples to the appropriate ranges.

4.3.4 Post-Processing

After generating the synthetic samples, the three-dimensional vectors are transformed into a pandas DataFrame. To mitigate potential fluctuations within the generated (x, y) values, we apply Exponential Moving Average (EMA) to each sample within the DataFrame as specified in 3.4.4. The DataFrame object provides the *ewm* function, which calculates the EMA for a given column in the DataFrame. It's crucial to note that we should not apply EMA across the entire DataFrame at once because each sample within the DataFrame is independent of the others. Therefore, we first need to extract a subset of the original DataFrame that contains a single sample, and then apply EMA to that subset.

4.3.5 Evaluating the Synthetic Data

As described in the previous chapter, both quantitative and qualitative measures are used to assess the quality of the synthetic data. To visualize the results, the graphical library *matplotlib* is used.

Length Distribution

To depict the sample length distribution of the real and synthetic data, we simply store each samples individual length by first extracting the padded sample from the DataFrame, then removing the rows with $gen = 0$. Because each row corresponds to a 10ms timestep, the time taken for the drag-and-drop sequence corresponds to the number of unpadded rows times 10ms.

Average Delta distance

To visualize the average delta distance to the target location, the distance vector of each sample has to be calculated and the added to a single result vector, which is divided by the number of unique samples within our dataset. The procedure is formally described in Algorithm 1, where each sample, which is originally padded is unpadded by excluding all rows with $gen = 0$. The target location, which we use to calculate the relative distance, is the last entry of our unpadded sample. The euclidean distance between the current (x_i, y_i) and the target coordinates are then calculated and stored as a distance vector. To maintain consistency in our dimensions, the distance vector is padded with 0 until the vector has the length of the longest occurring sequence in our dataset. The distance vector is then added to a result vector which stores the sum of all distance vectors. To attain the average distance vector, we divide the vector by the number of unique samples in our DataFrame.

Algorithm 1 Average delta distance to target

```
1: function AverageDeltaDistance($DataFrame \in \mathbb{R}^{n\times 3}, K, num\_samples$)
2:     $D \leftarrow 0$                                ▷ Initialize a $k \times 1$ vector $D$ with $0$
3:     for $sample \in DataFrame$ do
4:         $unpadded \leftarrow sample \backslash \{gen = 0\} \in \mathbb{R}^{S\times 2}$    ▷ Remove padding and drop "gen" column
5:         $x\_target \leftarrow unpadded[S][0]$
6:         $y\_target \leftarrow unpadded[S][1]$
7:         $distance \leftarrow \sqrt{\sum_{i=1}^{S}(unpadded[i][0] - x\_target)^2 + (unpadded[i][1] - y\_target)^2} \in \mathbb{R}^{S\times 1}$
8:         for $i = 1$ in $range(K - S)$ do              ▷ pad the distance vector
9:             $distance[S + i][0] = 0$
10:            $distance[S + i][1] = 0$
11:        end for
12:        $D \leftarrow D + distance$
13:    end for
14:    return $D \backslash num\_samples$          ▷ Return the average delta distance vector
15: end function
```

K-Nearest Neighbours

When calculating the k-nearest neighbors of a synthetic sample, we first attain an arbitrary sample within the synthetic DataFrame. The padded rows within the synthetic sample are dropped by removing all rows with $gen = 0$. The Euclidean distance vector to the target location is calculated with respect to the unpadded synthetic sample. To calculate the similarity of the synthetic distance vector and the distance vector of each real sample, we use DTW as described in section 3.5.1. To calculate the DTW distance between the samples, we use the *dtaidistance* library which provides the implementation of the DTW algorithm [39]. The calculated DTW distance, alongside the currently tested sample, is stored in a list that tracks the result of each sample within the DataFrame. After processing all samples, the resulting list is sorted based on the DTW distance, in ascending order. The first k-elements within the list corresponds to the samples with the smallest DTW distance, which is used to visualize the nearest neighbor of the provided synthetic sample. The formal algorithmic description of our implementation is described in Algorithm 2 and 3.

Algorithm 2 Calculate the k-nearest neighbours using DTW

```
1:  function KNN-DTW($sample \in \mathbb{R}^{S\times 2}, DataFrame \in \mathbb{R}^{N\times 3}$)
2:      $x\_target \leftarrow sample[S][0]$                                  ▷ Get Last x-coordinate
3:      $y\_target \leftarrow sample[S][1]$                                  ▷ Get Last y-coordinate
4:      $distance \leftarrow \sqrt{\sum_{i=1}^{S}(sample[i][0]-x\_target)^2+(sample[i][1]-y\_target)^2} \in \mathbb{R}^{S\times 1}$
5:      $result \leftarrow ()$                                               ▷ List for storing nearest neighbours
6:      for $sample\_real \in DataFrame$ do
7:          $unpadded \leftarrow sample\_real \backslash \{gen = 0\} \in \mathbb{R}^{K\times 2}$   ▷ Remove padding and drop ”gen” column
8:          $curr\_x\_target \leftarrow unpadded[K][0]$
9:          $curr\_y\_target \leftarrow unpadded[K][1]$
10:         $curr\_distance \leftarrow \sqrt{\sum_{i=1}^{K}(unpadded[i][0]-curr\_x\_target)^2+(unpadded[i][1]-curr\_y\_target)^2} \in \mathbb{R}^{K\times 1}$
11:         $dtw\_distance \leftarrow DTW(distance, curr\_distance)$
12:         $result \leftarrow dtw\_distance, unpadded$                     ▷ Store DTW-distance and the corresponding sample
13:     end for
14:     $Sort(result)$                                                      ▷ Sort by DTW distance in ascending order
         return $result[0:k]$
15: end function
```

Algorithm 3 Dynamic Time Warping

```
1:  function DTW($S \in \mathbb{R}^{n\times 1}, T \in \mathbb{R}^{m\times 1}$)
2:      Initialize a $(n+1)\times(m+1)$ matrix $D$ with $\infty$
3:      $D[0][0] \leftarrow 0$
4:      for $i = 1$ to $n$ do
5:          for $j = 1$ to $m$ do
6:              $cost \leftarrow d(s_i, t_j)$                                  ▷ Distance between points
7:              $D[i][j] \leftarrow cost + min\{D[i-1][j], D[i][j-1], D[i-1][j-1]\}$ ▷ Choose the path with minimum accumulated cost
8:          end for
9:      end for
10:     return $D[n][m]$                                                    ▷ Return the DTW distance
11: end function
```

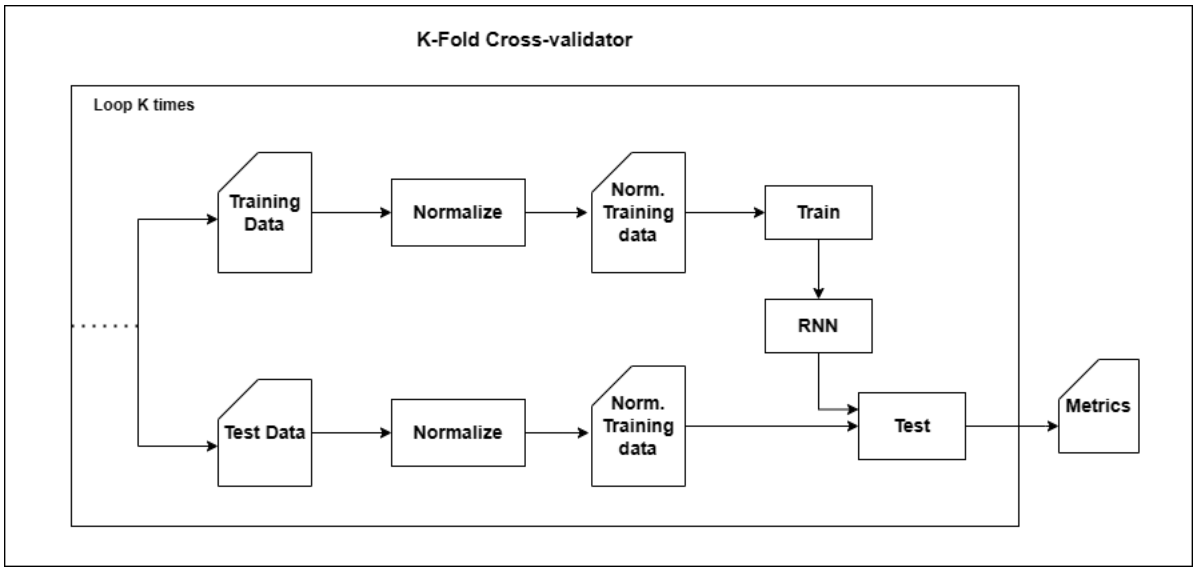


Figure 4.3.2: Cross-validation training of RNN model

Train on Synthetic, Test on Real

As depicted in figure 4.3.3, we take both the elderly and young population synthetic datasets and merge them into a single dataset. Before we merge the DataFrames, the DataFrame with the shortest $max\_sequence\_length$ has to be padded to the length of the $max\_sequence\_length$ in the other DataFrame. This step is necessary because a machine learning model can only be prompted with a certain predetermined input dimension. Alongside the merged DataFrame, we create a column vector that stores the values 0 and 1. The value 1 corresponds to a younger drag-and-drop sample and 0 for the elderly sample. Each sample within the merged DataFrame, X_i has a corresponding label y_i which is used to train an RNN model to distinguish the drag-and-drop sequences produced by an elderly or young person. The model is built using *keras*, which provides a high-level interface of the deep learning library *TensorFlow* [3]. As depicted in figure 4.3.2, the model is trained using a cross-validator. The purpose of the cross-validator is to provide an average estimation of our model's performance. At each iteration within the cross-validator, the DataFrame, alongside the labels is split into batches of shuffled data, where 80% of the data is used for training and 20% for testing. To preserve the ratio between the elderly and young samples within our training and testing data, we use the *StratifiedKFold* function provided by *sklearn*[33]. Before training the model, we first normalize the training samples and use the same normalization parameters to fit our testing data. It is important to not normalize the entire DataFrame before we split the data, as this can result in biased results. The recall, precision and F1 scores are stored in a list for each iteration and the average scores are calculated after finishing all iterations.

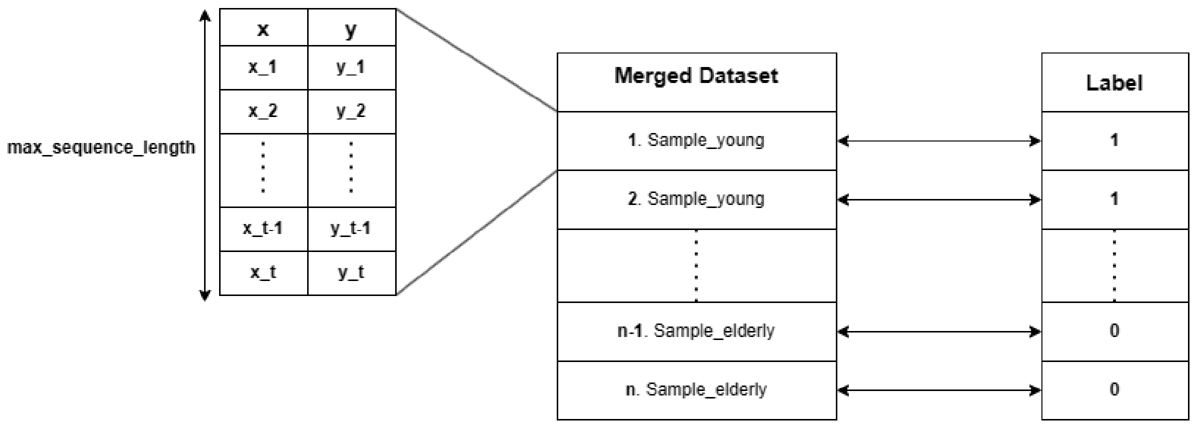


Figure 4.3.3: Merged DataFrame and its corresponding labels

4.4 Code Repository

The code for both the mobile application, as well as the synthetic data generation can be accessed through our public GitLab repository[1].

[1]https://git.cse.kau.se/laoajala100/MasterThesis.git

Chapter 5

Result and Discussion

This section includes the gathered results from both the data collection, and the experiments.

5.1 Data Collection

The data utilized in this experiment were collected at Karlstad University as well as other public areas throughout Karlstad.

Figure 5.1.1 presents the distribution of various groups within the collected dataset. The total number of participants was 34, with 19 representing the younger population and 15 for the elderly. The number of participants who preferred to use their thumb or finger to perform the experiment was equally distributed. Most participants were aged 17-45, representing 55.9% of all participants, while 44.1% were aged above 45. It should be noted that the dataset does not include any participants aged 0-17, hence their absence from the dataset. The majority of participants were right-handed, making up 94.1% of the dataset, while the remaining 5.9% were left-handed.

Figure 5.1.2 provides a boxplot of the individual length of each drag-and-drop sequence within both the elderly and young datasets. The outliers, represented by the circles, were removed from the datasets as described in section 3.4.1. After the outliers were removed, the average length of the time taken to complete the drag-and-drop sequence was 1.22s for the younger population and 2.64s for the elderly population. The standard deviation for the elderly was 0.56s and 0.48s for the younger group. The longest recorded sample within the elderly population was 4.10s and 2.66s for the younger population. The shortest sample for the elderly population was 1.33s and 0.51s for the younger population.

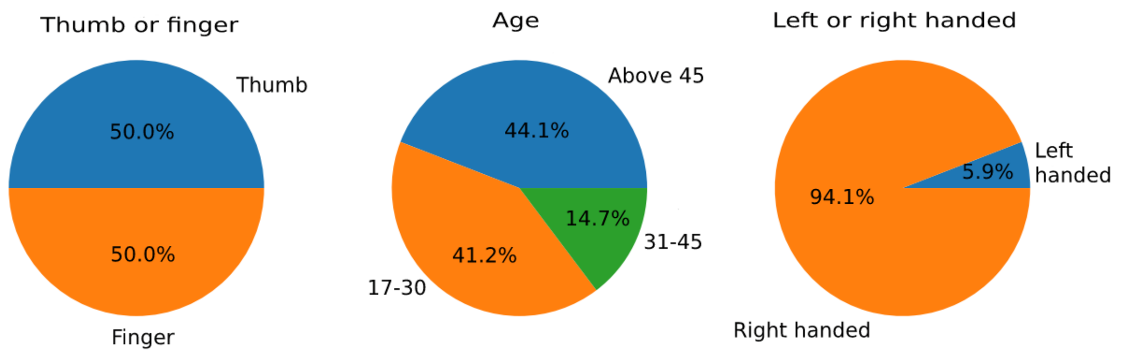


Figure 5.1.1: Result from questionnaire regarding the participant information

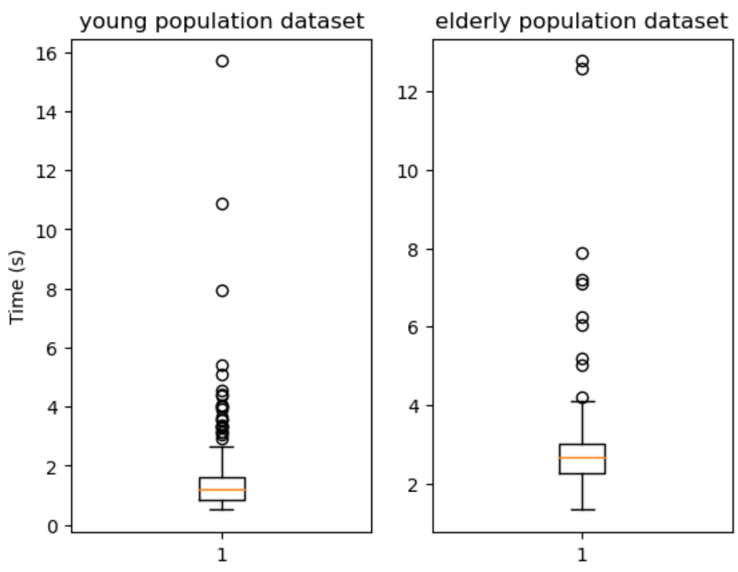


Figure 5.1.2: Boxplot displaying the per sample length of both the young and elderly population

5.2 GAN Parameters

The hyperparameters used of the final GANs used to generate the data to answer **RQ1** and **RQ2** can be found in figure A.0.1 in the appendix.

5.3 RQ1: Fidelity of the Synthetic Data

In this section, we present the results to answer the research question RQ1, which was formulated as *How well does synthetic data resemble UI interaction data for different user groups?*. Figure 5.3.1 display the per-sample length distribution of the real and synthetic datasets. The GAN trained on the younger population dataset shows a similar pattern to the original dataset; both distributions have more occurrences of samples within the 0.5-1.75s range and fewer samples when the drag sequence exceeds a length >1.75s. The GAN trained on the elderly dataset also shows a similar structure in the overall per-sample distribution but more bias toward samples with lengths around 3s. Table 5.3.1 shows the mean, median, and standard deviation of the real and synthetic datasets. The mean, median, and deviation of the younger datasets show very little difference, which supports the similarity displayed in the distribution plots. The mean of the elderly and synthetic dataset does, however, differ with 83ms which is expected as the distribution plots show a similar trend within the synthetic dataset.

Figure 5.3.2 shows the average delta distance to the target location of each sample for both the younger and elderly datasets. The synthetic data of the younger and elderly population shows an almost identical decrease in delta distance to the target as time increases. This indicates that the synthetic samples do, on average, incorporate the temporal correlations between the (x, y) coordinates within each sample. Both the elderly and younger synthetic data do, however, display a small increase of delta distance in the early stages of the drag-and-drop. The elderly synthetic data shows a small offset in divergence compared to the actual dataset. This is to expect as the prior analysis regarding the per-sample distribution shows that the elderly synthetic data has, on average longer samples.

In order to verify whether the synthetic data generated by both GANs has not memorized the original dataset, we select three random samples from the synthetic datasets and retrieve their nearest neighbor within the original dataset. Figure 5.3.3 and 5.3.4 depict the resulting nearest neighbor of the younger and elderly population synthetic data. We see that the synthetic samples from both population groups follow a similar trend observable in the original datasets. The samples do, however, differ in length when compared to their nearest neighbors, which indicates that the synthetic samples are not present in the original dataset. We can also observe that the first and third synthetic samples within the younger population contain noise in the beginning stages of the sequences, indicated by the sudden increase/decrease in the delta distance to the target location. Similarly, for the third elderly synthetic sample, we observe a slight sudden change in the delta distance at the beginning of the sequence.

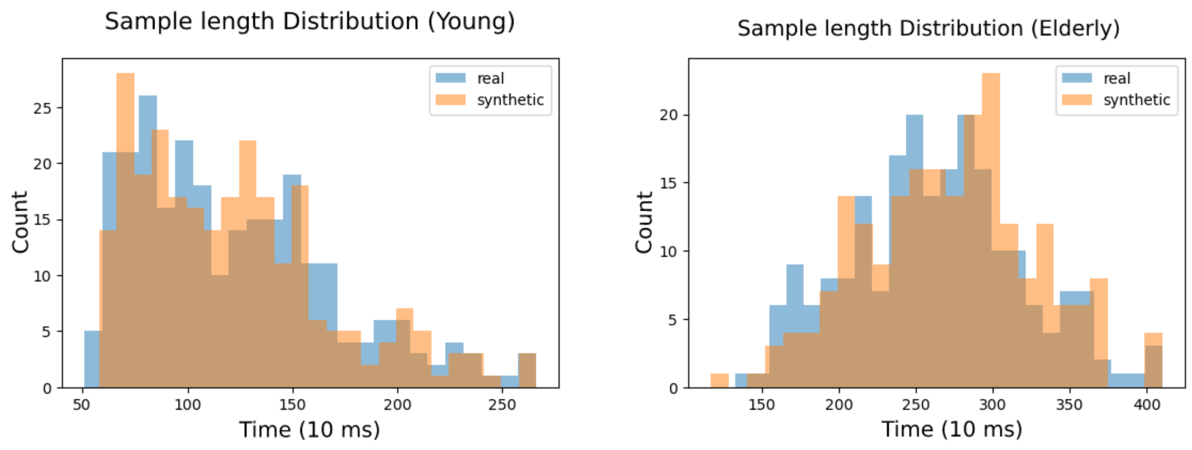


Figure 5.3.1: Sample length distribution for the real and synthetic datasets

Dataset	Mean (s)	Median (s)	Std. Dev. (+/- s)
Real (Younger)	1.224	1.130	0.476
Synthetic (Younger)	1.218	1.160	0.425
Real (Elderly)	2.634	2.635	0.562
Synthetic (Elderly)	2.717	2.745	0.559

Table 5.3.1: Mean and standard deviation of the sample length distribution

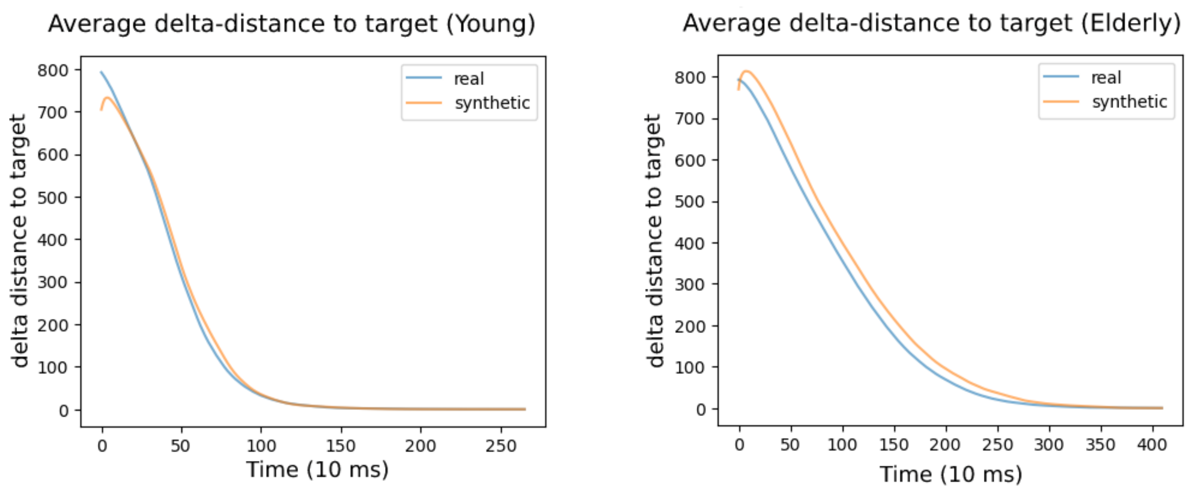


Figure 5.3.2: Average delta distance to target location

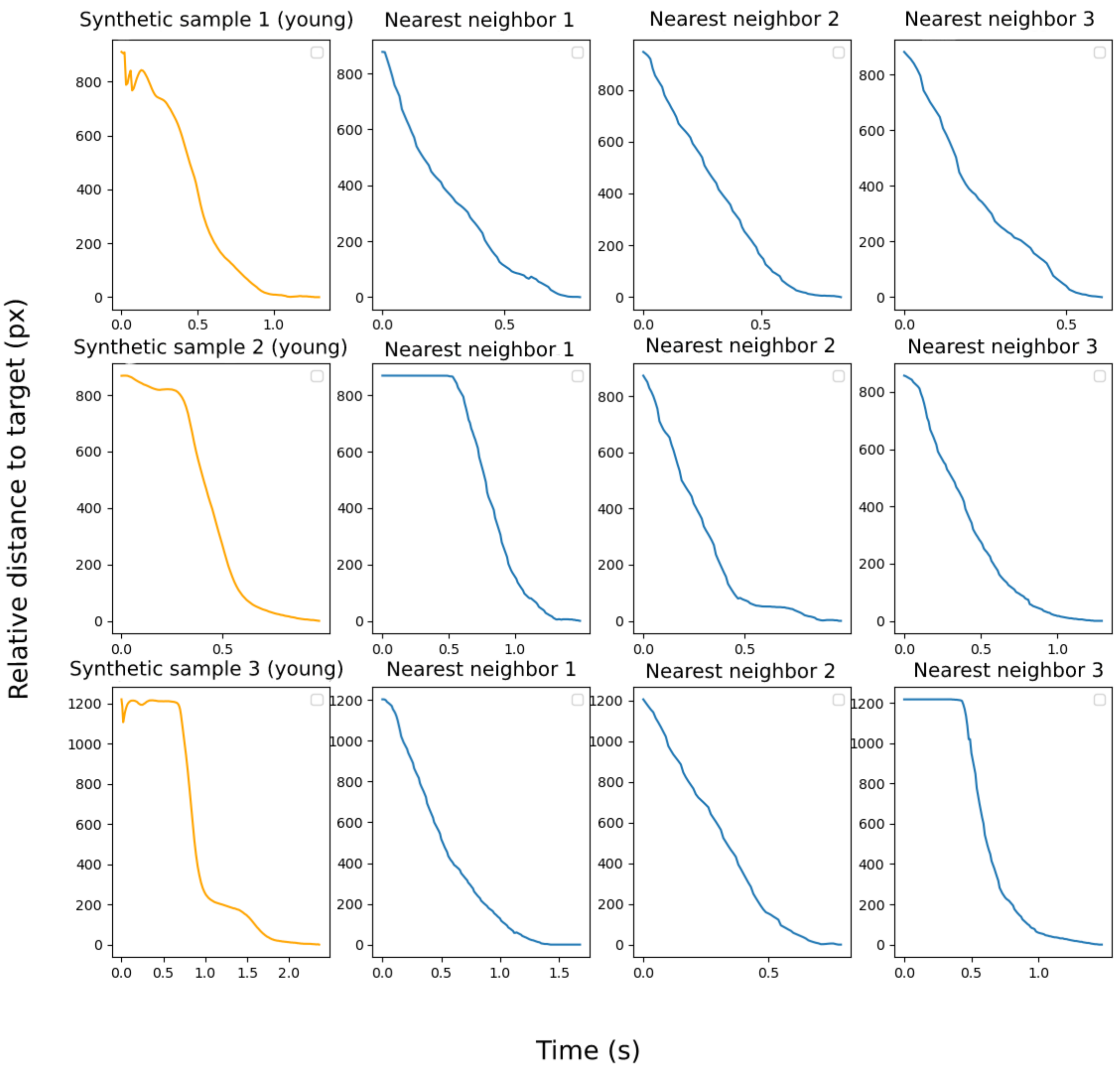


Figure 5.3.3: Nearest neighbors of the synthetic samples (Young)

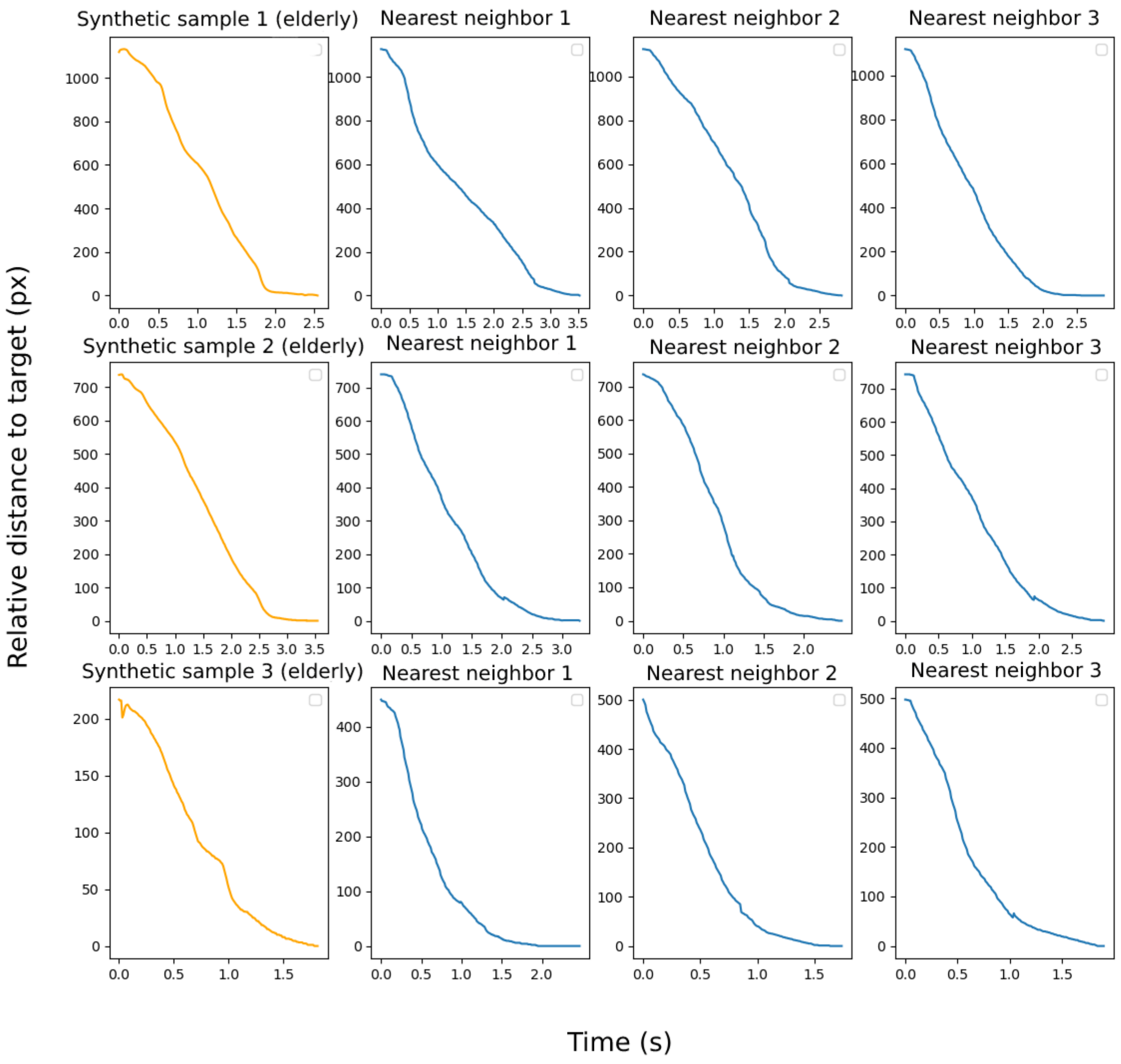


Figure 5.3.4: Nearest neighbors of the synthetic samples (Elderly)

5.4 RQ2: Performance Evaluation

In this section, we present the results to answer the research question RQ2, which was formulated as *How well does the synthetic UI interaction perform compared to real user interaction data?*. As discussed in previous sections, we deploy an external validator, in this case a simple RNN architecture to classify whether the drag and drop sequences belong to the younger or elderly populations. The model used to grant the upcoming result is displayed in figure A.0.1, present in the appendix. The model is trained for 50 epochs, and a batch size of 64 is used. The total # of samples amounts to 477, where 382 were used for training and 95 for evaluating the model. Out of the 382 samples, 210 corresponds to the younger population and 172 to the elderly population. The same proportions are applied to the testing dataset. Figure 5.4.1 depicts each iteration's resulting precision, recall and F1 scores within the cross-validation. We observe that the model performs, on average, better across all 3 metrics when trained on the real dataset. The synthetic data does, however, score slightly higher in some iterations, but the difference is close to negligible. Table 5.4.1 shows the average scores of both the synthetic and real datasets. Both datasets exhibit high recall scores, where the real dataset edges the synthetic dataset with 2.4%. The variance in recall is also small in both cases, which indicates that the mean recall is a reasonable estimate for both datasets. The average precision score between both datasets differs with 1.9%, where the model trained on the real dataset slightly outperforms the model trained on the synthetic data. The average F1 score follows a similar trend as with the other metrics.

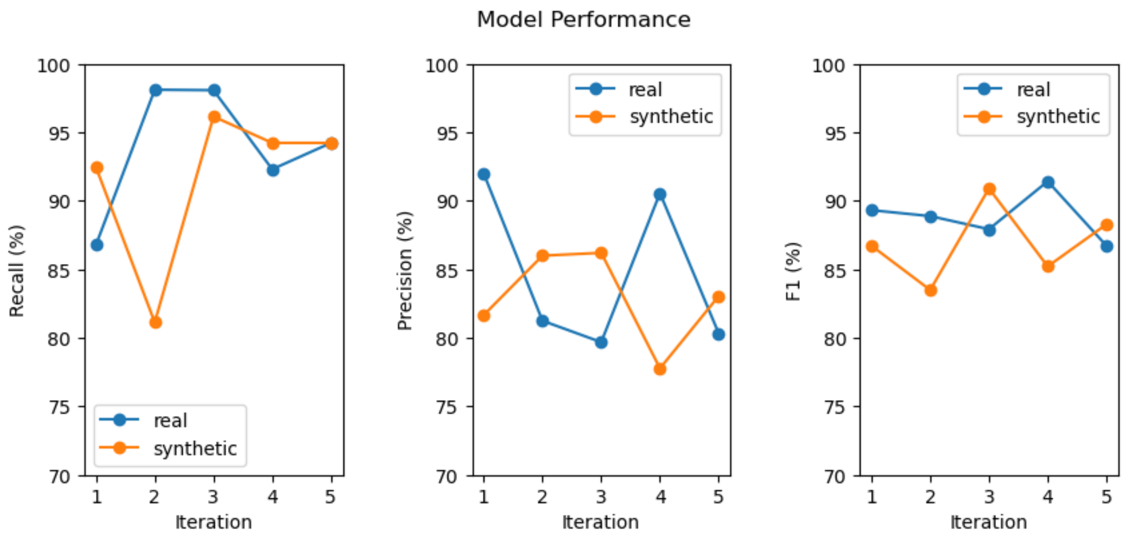


Figure 5.4.1: Model performance when trained and tested on both real and synthetic data

Metrics	Real	Synthetic
Recall %	**94.0**	91.6
Std. Dev. (Rec.)%	**4.2**	5.4
Precision%	**84.8**	82.9
Std. Dev. (Prec.)%	5.4	**3.1**
F1%	**88.9**	86.9
Std. Dev. (F1)%	**1.6**	2.5

Table 5.4.1: Average model performance for both synthetic and real datasets

5.5 Discussion

5.5.1 Fidelity of the Synthetic Data

This section aims to answer the research question RQ1, which asks the following: *How well does synthetic data resemble UI interaction data for different user groups?*. The presented result indicates that the elderly and younger population GANs produce similar per-sample lengths as the real dataset. This indicates that the GANs produce diversity regarding the individual sample length, but also carrying similar statistical properties as in the real dataset, where the mean, median and standard deviation are similar across the datasets. This property is important because, even if we succeed in generating high quality samples, but the diversity is bad, the synthetic data will not reflect the properties of all captured use cases in our original dataset. More formally, we can say that both GANs do not suffer from mode-collapse regarding the length distribution. As mentioned in chapter 3, mode-collapse is the result when we have a GAN that generates same samples for different input noise vectors. The length distribution does however not suffice to assure whether the drag and drop sequences follows similar trends in temporal correlation between the data-points.

Figure 5.3.2 shows the average delta distance to the target location for both synthetic datasets. We can observe that the synthetic dataset representing the young population follows a similar pattern in decreasing delta distance as time increases. This indicates that the synthetic dataset carries temporal correlation within each sample, similar to the real dataset. This result is important because the temporal correlation of our data-points carries the information about the actual drag and drop sequences. We can also observe that the average starting distance for the younger synthetic dataset is closer to the target location than that of the real dataset. This implies that the younger population GAN generates samples where the starting and drop location are closer than that of the real dataset. The reason behind this can be that the *min-max generator*, discussed in 3.4.2, has not fully learned the per-sample ranges of the real dataset. The elderly population synthetic data shows similar patterns regarding the decreasing delta distance but a slight increase in convergence to the target location. The most likely explanation for this is the over representation of longer sequences within the synthetic dataset, compared to the real dataset. This implies that the elderly synthetic dataset may contain a slight bias towards longer sequences which can affect the possible use-cases where we might use our synthetic data. In both the elderly and younger synthetic datasets, a small increase and decrease of the average delta distance appears in the beginning of the sequence. This behaviour appears for both GANs, which are trained independently. The reason behind this behaviour is unclear, but is most likely a architecture problem as the result appears for both GANs. Thus, a possible solution is to experiment with the settings within the GAN, e.g. adding more complexity to the architecture by increasing the number of layers and units within each model.

The synthetic datasets across both the young and elderly population shows signs of both diversity and uniqueness when depicting the synthetic samples nearest neighbours. None of the synthetic samples depicted in 5.3.3 and 5.3.4 has an identical neighbour. This is important because we can use the synthetic dataset to augment an existing dataset with limited samples without introducing duplicate entries.

5.5.2 Performance of the Synthetic Data

This section aims to answer the research question RQ2, which asks the following: *How well does the synthetic UI interaction perform compared to real user interaction data?*.

As expected, the external RNN-model trained on real data outperforms the synthetic data across all metrics, on average. This result complies with prior research that use a similar approach when evaluating the quality of their synthetic data [6, 27, 41], which is to expect because the quality of our real dataset contains less noise than that of the synthetic dataset. The difference in performance across all metrics is however small, which further highlights that the quality of our synthetic data, generated by both GANs, follows similar levels to that of the real UI interactions. It is worth mentioning that the model architecture used for this particular classification task may not be the most optimal solution, and improvements regarding hyperparameter tuning and adding more complexity to the model could possibly yield better performances. However, we still believe that the same trend should emerge despite possible model improvements, where the real dataset outperforms the synthetic data, but with an possible slimmer margin regarding the performance results.

How the RNN model learns to distinguish between the elderly and younger drag and drops is also an important aspect to consider when making assessment regarding the quality of our synthetic. We believe that there are 2 central patterns within the drag and drop sequence that may affect the result of our RNN-model. First, we can observe that the average length differences between the elderly and young populations samples, as depicted in table 5.3.1, are quite different, which may affect how the RNN-model learns to distinguish between the elderly and young populations. The second possibility is that the model learns to distinguish between the samples by learning different patterns in temporal correlations of the captured coordinates. Recall that we capture coordinates, starting when the user begins the drag operation and finish when the square is dropped in the target location. During this time period, the user may possibly drop the square outside the target location or hover the square around the target location before dropping. It is observed that the elderly population tends to focus more on precision than speed, which is the opposite for the younger population [32, 38]. Thus, we believe that the elderly drag and drop sequences contains more clustering of similar coordinates towards the end of the sequence, mainly because the elderly aims to drop the square precisely in the middle of the drop location. This can also be partially observed figure 5.3.4 and figure 5.3.3, where we see the nearest neighbours of our synthetic samples. The elderly samples, both real and synthetic, tend to plateau around 0 for a longer time than for the younger samples, which implicitly means that the square is hovering of the target location for a longer time.

The fact that the GANs produced synthetic data which resulted in a performance level close to that of the real data further shows the value of using GANs for certain user interaction tasks. This is particularly promising in tasks where the collection of user data is limited, costly or difficult to collect, such as with certain hard-to-access user groups.

Although we showed that the synthetic data yields similar results to that of the real dataset when used for a simple classification task, there is still uncertainty regarding how well the synthetic data performs in other more complex tasks, which is out of scope in this thesis but should definitely be examined to further validate the possible use-cases of the synthetic data.

5.5.3 Threats to Validity

There are several parts within the project that may affect the validity of our gathered results. We discuss 3 types of validity threats, internal validity, external validity and construct validity. Internal validity is the degree to which the result of our study is caused by the observed independent variables, and not the possible flaws in our experimental design. External validity refers to how general the result of our study is, and if the result applies to other settings. Construct validity refers to whether one can justifiably claim whether the measurement used in the study reflects the concepts which they claim to measure.

Internal Validity

The study uses an external RNN to quantitatively measure the performance of our synthetic data compared to the real data when used for a classification task. The attained results indicates that the synthetic data slightly reduces the performance of our RNN compared when using the real dataset. Although we use a cross-validator to estimate how the model would perform in practice, the optimizer used to train the RNN is of stochastic nature, meaning that the granted result may vary based on how good the model converge towards the global minima. Furthermore, we see in figure 5.4.1

that some iterations within the cross-validator produces quite large variations across both the recall and precision scores, which can be caused by the used optimizer and not the quality of our data.

External Validity

The gathered data resembles 2 groups, the elderly and the younger population. This separation of groups is only based on the age factor which is most likely not enough to generalize across all possible subgroups of people. For instance, the dataset do not consider whether the user has any medical conditions that may affect how they interact with an UI. Thus, our results should not be interpret as a grand solution for all possible user groups. Also, the generated synthetic data may not capture all variations of the user interaction data and may not fully apply to real-world scenarios. Another important consideration is that our result is solely based of a single UI gesture, namely the drag and drop, which is one of many existing gestures one could experiment with. The setting used to collect user data does also not reflect a real-world scenario, where we randomly displace the drag and drops in each iteration within the UI.

Construct Validity

We use the delta distance to the target location when validating the temporal correlation of the samples. This only displays the magnitude of the distance at each time step, but not the direction of our drag sequence. Hence, we can only assure the quality of the temporal correlation but not the diversity regarding the direction of our drag and drop sequence. Hence, the synthetic samples may include a majority of drag and drop sequences that follows a similar pattern, e.g. only upward dragging, but differs in the start and drop location. Furthermore, we calculated the nearest-neighbours to determine whether the synthetic data contains diversity and uniqueness. Only three samples within the synthetic datasets where depicted, which may not be enough to claim diversity and uniqueness when considering the complete dataset. Another aspect to consider is how we evaluate the performance of our synthetic data. The classification model may only learn the length distribution during training and not consider the actual temporal correlation within each sample when classifying whether a drag and drop sequence comes from the elderly or young population. This may imply that that, even if our samples contains illogical sequences, the model may still distinguish the samples based on its length. Also, the use of a classification task may not be sufficient to assert that the performance of synthetic data applies to other problem settings, e.g. predictive modelling where we might want to predict where the user might drop the the square, given a subsequent of the complete operation.

5.5.4 Limitations

One limitation of this project is that only a single UI gesture, the drag and drop, were used for analysis. Using a single UI gesture minimized the time needed for the participant to perform the experiment. Suppose a more comprehensive set of UI gestures were to be tested. In that case, data collection becomes more tedious and time-consuming because fewer people would likely participate in the experiment. Additionally, the scenario used to capture the drag-and-drop sequences does not reflect a real-world scenario where one might test a proposed UI design on multiple participants. Furthermore, we only utilize a single GAN architecture, namely the doppelGANger architecture. Testing other GAN architectures [6, 41] might provide better results within this problem set. Furthermore, we did not incorporate the metadata of each drag-and-drop sample when training the GAN network. This is a powerful feature within the doppelGANger model that can be used to characterize each sample, e.g., whether the sample was performed by a right or left-handed person or the particular age of the participant.

Another major limitation was the limited hardware resources used to train the GAN network. Even with our small dataset, training the GAN took approximately 2 hours. This makes tuning the hyperparameters within the GAN tedious and limited, which may lead to a model that may produce less optimal results.

Moreover, the study focuses on user interactions that can easily be transferred to a time series representation of the UI interaction, which can be used for training the GANs. In general, other types of UI interactions or problem settings may not be as easily adapted to this particular approach.

Chapter 6

Conclusions and Future Work

This thesis aims to bridge the gaps between the use of GANs in the domain of usability testing, where we mainly focus on the drag-and-drop operation. Moreover, the research aims to provide valuable insights regarding the possibilities and limitations of modeling synthetic user interactions within a UI.

The results of our study indicate, both qualitatively and quantitatively, that using the doppelGANger architecture for the purpose of generating UI gesture data, provides high-quality synthetic data. Both generators of the elderly and younger population have not only learned to capture the statistical properties within the original dataset but also produce samples that have diversity and follow similar temporal correlation patterns as the real drag-and-drop sequences. Furthermore, we have shown that the synthetic data is highly competitive, compared to the real dataset, when used in a classification task where an RNN model is trained to distinguish between the elderly and younger drag-and-drop sequences.

By demonstrating that the GAN models can generate high-quality synthetic data which closely mimics real user interactions of the drag-and-drop operation, our study offers new possibilities for enhancing current usability testing, where the data collection might be limited. Not only does this open up new ways for creating diverse and statistically consistent data for training RNN models in a classification task, but it also has implications for designing and developing more user-friendly interfaces. For instance, the ability to accurately model and generate different user interaction patterns, such as those of elderly and younger populations, can help designers anticipate user behaviors and needs, ultimately leading to the creation of more inclusive and accessible software systems. This could improve the way usability testing is conducted in the future, where less time has to be spent on extensive data collection, but also mitigate potential hurdles when certain user groups may be hard to access.

6.1 Future Work

This thesis has only graced the potential of generating synthetic user interaction data. Including a wider set of UI gestures, such as pinching and tapping, would provide valuable insights regarding the strengths and weaknesses of GANs within these particular settings, which further generalizes the results. Furthermore, it would be interesting to investigate whether a single GAN model could be used to model multiple user groups simultaneously and achieve high-quality synthetic data. This would provide a centralized solution which becomes more manageable to maintain and also easier to deploy in practice. Another interesting area is the choice of generative model. One could perform a comparative study between multiple GAN architectures, such as TimeGAN[41] and RCGAN[6] to further strengthen the potential options and benefits of different GAN architectures within this specific domain.

Bibliography

[1] Agarap, Abien Fred. “Deep learning using rectified linear units (relu)”. In: *arXiv preprint arXiv:1803.08375* (2018).

[2] Arjovsky, Martin, Chintala, Soumith, and Bottou, Léon. *Wasserstein GAN*. 2017. arXiv: `1701.07875 [stat.ML]`.

[3] Chollet, François et al. *Keras*. `https://keras.io`. 2015.

[4] Dehlinger, Josh and Dixon, Jeremy. “Mobile application software engineering: Challenges and research directions”. In: *Workshop on mobile software engineering*. Vol. 2. 2011, pp. 29–32.

[5] *Docker*. en. May 2023. url: `https://docs.docker.com/get-started/` (visited on 05/12/2023).

[6] Esteban, Cristóbal, Hyland, Stephanie L., and Rätsch, Gunnar. *Real-valued (Medical) Time Series Generation with Recurrent Conditional GANs*. 2017. arXiv: `1706.02633 [stat.ML]`.

[7] Esteban, Cristóbal, Hyland, Stephanie L., and Rätsch, Gunnar. *Real-valued (Medical) Time Series Generation with Recurrent Conditional GANs*. 2017. arXiv: `1706.02633 [stat.ML]`.

[8] *General Data Protection Regulation (GDPR) – Official Legal Text*. url: `https://gdpr-info.eu/` (visited on 05/31/2023).

[9] *GitHub - gretelai/gretel-synthetics: Synthetic data generators for structured and unstructured text, featuring differentially private learning*. url: `https://github.com/gretelai/gretel-synthetics` (visited on 05/12/2023).

[10] *Global: number of smartphone users 2013-2028 | Statista*. url: `https://www.statista.com/forecasts/1143723/smartphone-users-in-the-world` (visited on 05/30/2023).

[11] Goodfellow, Ian. *NIPS 2016 Tutorial: Generative Adversarial Networks*. 2017. arXiv: `1701.00160 [cs.LG]`.

[12] Goodfellow, Ian, Bengio, Yoshua, and Courville, Aaron. *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press, 2016.

[13] Goodfellow, Ian J., Pouget-Abadie, Jean, Mirza, Mehdi, Xu, Bing, Warde-Farley, David, Ozair, Sherjil, Courville, Aaron, and Bengio, Yoshua. *Generative Adversarial Networks*. 2014. arXiv: `1406.2661 [stat.ML]`.

[14] Gulrajani, Ishaan, Ahmed, Faruk, Arjovsky, Martin, Dumoulin, Vincent, and Courville, Aaron. *Improved Training of Wasserstein GANs*. 2017. arXiv: 1704.00028 [cs.LG].

[15] Harshvardhan, GM, Gourisaria, Mahendra Kumar, Pandey, Manjusha, and Rautaray, Siddharth Swarup. “A comprehensive survey and analysis of generative models in machine learning”. In: *Computer Science Review* 38 (2020), p. 100285.

[16] Hochreiter, Sepp and Schmidhuber, Jürgen. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[17] *Human Interface Guidelines | Apple Developer Documentation*. url: https://developer.apple.com/design/human-interface-guidelines (visited on 05/30/2023).

[18] Janocha, Katarzyna and Czarnecki, Wojciech Marian. “On loss functions for deep neural networks in classification”. In: *arXiv preprint arXiv:1702.05659* (2017).

[19] Jobin, Anna, Ienca, Marcello, and Vayena, Effy. “The global landscape of AI ethics guidelines”. In: *Nature Machine Intelligence* 1.9 (2019), pp. 389–399.

[20] Jordon, James, Szpruch, Lukasz, Houssiau, Florimond, Bottarelli, Mirko, Cherubin, Giovanni, Maple, Carsten, Cohen, Samuel N., and Weller, Adrian. *Synthetic Data – what, why and how?* 2022. arXiv: 2205.03257 [cs.LG].

[21] Kaasila, Jouko, Ferreira, Denzil, Kostakos, Vassilis, and Ojala, Timo. “Testdroid: automated remote UI testing on Android”. In: *Proceedings of the 11th International Conference on Mobile and Ubiquitous Multimedia*. 2012, pp. 1–4.

[22] Kingma, Diederik P. and Ba, Jimmy. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[23] Kobayashi, Masatomo, Hiyama, Atsushi, Miura, Takahiro, Asakawa, Chieko, Hirose, Michitaka, and Ifukube, Tohru. “Elderly User Evaluation of Mobile Touchscreen Interactions”. In: *Human-Computer Interaction – INTERACT 2011*. Ed. by Pedro Campos, Nicholas Graham, Joaquim Jorge, Nuno Nunes, Philippe Palanque, and Marco Winckler. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 83–99. isbn: 978-3-642-23774-4.

[24] Lawrence, Jeannette. *Introduction to neural networks*. California Scientific Software, 1993.

[25] Leach, Paul J., Salz, Rich, and Mealling, Michael H. *A Universally Unique IDentifier (UUID) URN Namespace*. RFC 4122. July 2005. doi: 10.17487/RFC4122. url: https://www.rfc-editor.org/info/rfc4122.

[26] Leporini, Barbara, Buzzi, Maria Claudia, and Buzzi, Marina. “Interacting with mobile devices via VoiceOver: usability and accessibility issues”. In: *Proceedings of the 24th Australian computer-human interaction conference*. 2012, pp. 339–348.

[27] Lin, Zinan, Jain, Alankar, Wang, Chen, Fanti, Giulia, and Sekar, Vyas. "Using GANs for Sharing Networked Time Series Data". In: *Proceedings of the ACM Internet Measurement Conference*. ACM, Oct. 2020. doi: `10.1145/3419394.3423643`. url: `https://doi.org/10.1145%5C%2F3419394.3423643`.

[28] Macias, Elsa, Suarez, Alvaro, and Lloret, Jaime. "Mobile sensing systems". In: *Sensors* 13.12 (2013), pp. 17292–17321.

[29] *Material Design for Android | Android Developers*. url: `https://developer.android.com/develop/ui/views/theming/look-and-feel` (visited on 05/30/2023).

[30] McKinney, Wes. "Data Structures for Statistical Computing in Python". In: *Proceedings of the 9th Python in Science Conference*. Ed. by Stéfan van der Walt and Jarrod Millman. 2010, pp. 51–56.

[31] Nayebi, Fatih, Desharnais, Jean-Marc, and Abran, Alain. "The state of the art of mobile application usability evaluation". In: *2012 25th IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*. 2012, pp. 1–4. doi: `10.1109/CCECE.2012.6334930`.

[32] Nurgalieva, Leysan, Jara Laconich, Juan José, Baez, Marcos, Casati, Fabio, and Marchese, Maurizio. "A Systematic Literature Review of Research-Derived Touchscreen Design Guidelines for Older Adults". In: *IEEE Access* 7 (2019), pp. 22035–22058. doi: `10.1109/ACCESS.2019.2898467`.

[33] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[34] Ruiz, Jaime, Li, Yang, and Lank, Edward. "User-defined motion gestures for mobile interaction". In: *Proceedings of the SIGCHI conference on human factors in computing systems*. 2011, pp. 197–206.

[35] Sherstinsky, Alex. "Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network". In: *Physica D: Nonlinear Phenomena* 404 (2020), p. 132306.

[36] Singh, Dalwinder and Singh, Birmohan. "Investigating the impact of data normalization on classification performance". In: *Applied Soft Computing* 97 (2020), p. 105524.

[37] Tavenard, Romain. *An introduction to Dynamic Time Warping*. `https://rtavenar.github.io/blog/dtw.html`. 2021.

[38] Tsai, Tsai-Hsuan, Tseng, Kevin C., and Chang, Yung-Sheng. “Testing the usability of smartphone surface gestures on different sizes of smartphones by different age groups of users”. In: *Computers in Human Behavior* 75 (2017), pp. 103–116. issn: 0747-5632. doi: https://doi.org/10.1016/j.chb.2017.05.013. url: https://www.sciencedirect.com/science/article/pii/S0747563217303254.

[39] wannesm, khendrickx, Yurtman, Aras, Robberechts, Pieter, Vohl, Dany, Ma, Eric, Verbruggen, Gust, Rossi, Marco, Shaikh, Mazhar, Yasirroni, Muhammad, Todd, Zieliński, Wojciech, Craenendonck, Toon Van, and Wu, Sai. *wannesm/dtaidistance: v2.3.5*. Version v2.3.5. Jan. 2022. doi: 10.5281/zenodo.5901139. url: https://doi.org/10.5281/zenodo.5901139.

[40] Wobbrock, Jacob O., Morris, Meredith Ringel, and Wilson, Andrew D. “User-Defined Gestures for Surface Computing”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI ’09. Boston, MA, USA: Association for Computing Machinery, 2009, pp. 1083–1092. isbn: 9781605582467. doi: 10.1145/1518701.1518866. url: https://doi.org/10.1145/1518701.1518866.

[41] Yoon, Jinsung, Jarrett, Daniel, and Schaar, Mihaela van der. “Time-series Generative Adversarial Networks”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, and R. Garnett. Vol. 32. Curran Associates, Inc., 2019. url: https://proceedings.neurips.cc/paper_files/paper/2019/file/c9efe5f26cd17ba6216bbe2a7d26d490-Paper.pdf.

[42] Zhang, Dongsong and Adipat, Boonlit. “Challenges, methodologies, and issues in the usability testing of mobile applications”. In: *International journal of human-computer interaction* 18.3 (2005), pp. 293–308.

Appendix - Contents

Appendix A

Appendix

Settings	Elderly population GAN	Young population GAN
Epochs	25000	30000
Batch size	215	262
Max sequence	410	266
batch generation	1	1
Discriminator Rounds	2	2
Generator Rounds	1	1
noise vector size (Attribute generator)	10	10
noise vector size (Time series generator)	10	10
MLP layers (Attribute generator)	3	3
MLP units (Attribute generator)	100	100
RNN (LSTM) layers (Time series generator)	2	2
RNN (LSTM) units (Time series generator)	150	150
MLP layers (Discriminator)	5	5
MLP units (Discriminator)	200	200
MLP layers (Auxiliary discriminator)	5	5
MLP units (Auxiliary discriminator)	200	200
Loss function	Wasserstein-1	Wasserstein-1
Optimizer (Both discriminators)	Adam	Adam
Learning rate (Generator)	0.0005	0.0005
Learning rate (Discriminator)	0.0005	0.0003

Table A.0.1: GAN architectural setting for both younger and elderly population. Various parameters which has not been altered are not included in the table and instead uses the baseline settings provided in [9]

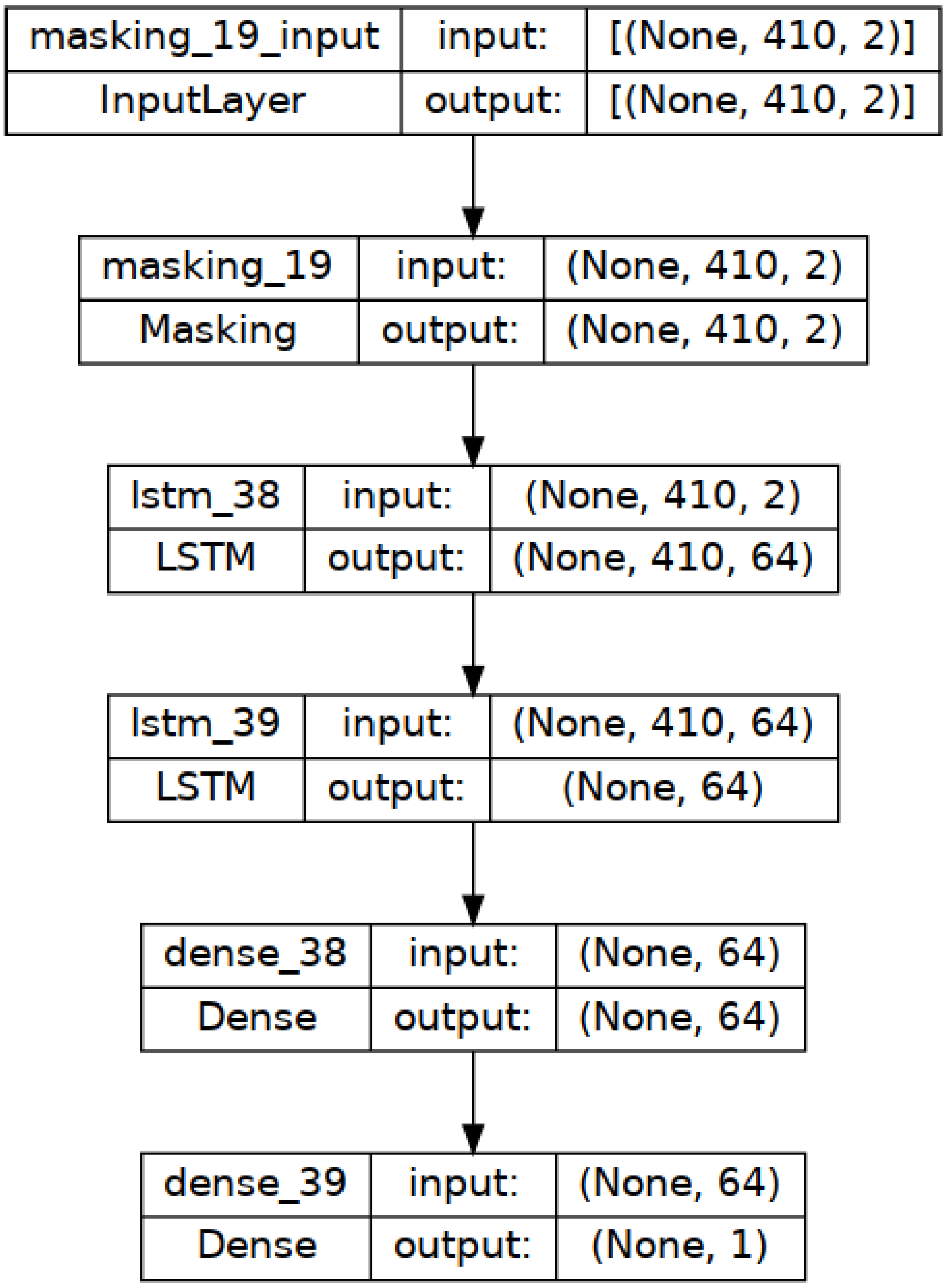


Figure A.0.1: Model used for the TSTR