POZNAN UNIVERSITY OF TECHNOLOGY

FACULTY OF COMPUTING AND TELECOMMUNICATION
Institute of Computing Science

Bachelor's thesis

BIOMETRIC IDENTIFICATION OF A SMARTPHONE USER USING GRAPH NEURAL NETWORKS

Jakub Grabowski, 151825
Filip Kozłowski, 151823
Krzysztof Matyla, 151778
Igor Warszawski, 151585

Supervisor
dr hab. inż. Szymon Szczęsny, prof. PP

POZNAŃ 2025

Tutaj będzie karta pracy dyplomowej;
oryginał wstawiamy do wersji dla archiwum PP, w pozostałych kopiach wstawiamy ksero.

Contents

Chapter 1

Introduction

Biometric data is a widely used – especially on mobile devices – for user authentication. It is also used for person recognition. As of 2024, most smartphones have biometric sensors, such as fingerprint readers (FIX SOURCE). Many computers also provide biometric authentication via face recognition (FIX SOURCE). These are, however, not the only possible recognition or authentication methods that use biometric data.

The project aimed to develop a model, along with a corresponding mobile app, capable of recognizing the user by their biometric data contained mostly within the keystroke data. The users in the study, which was a part of the project, provided their data by entering long stretches of text as testing data. Models were created for each user, with the standard model testing procedures and validations. A subgroup of the study participants was also asked to verify the model in real-life testing by writing short paragraphs in the application, which were sent to the server for user verification.

The scope of the work was to create a mobile application capable of gathering the keystroke data, which could then be used by the server to create Graph Neural Network (GNN) models tasked with recognizing the user as opposed to other possible users. Also in the scope was performing a study on a group of participants who provided the data for the project and participated in the application and model demonstration and testing.

The sources used in this thesis mostly concerned the two following groups: studies of keystroke data models and their effectiveness and the specialist literature on the topic of Graph Neural Networks.

The thesis has the following structure: Chapter 2 consists of some theory concerning biometrics, especially in the context of user input data, with a small literature review about using biometrics for user recognition. Chapter 3 contains basic theoretics about Graph Convolutional Networks, which are used for user recognition in the model created for the project. Chapter 4 is a brief overview of the project, explaining its components and the relationships between them. It includes the following sections: Section 4.1 consists of the description of the server. Section 4.2 describes the mobile application used for user data collection and model validation. Section 4.3 contains a description of the Neural Network model used for user recognition, complete with the hyperparameters used in model training and validation. Section 4.4 describes the feature selection used for a model. Section 4.5 discusses the metrics used in the model testing on data gathered from users and the testing results. Section 4.6 concerns the user testing with the help of study participants and the study results. Chapter 5 is a conclusion to the thesis.

Work on this project was divided as follows: Jakub Grabowski created the mobile application, set up and coordinated the project, and researched biometrics for his thesis paper. Filip Kozłowski

created the server and integrated the GNN model with it. He also planned and implemented communication between the server and the application. Krzysztof Matyla helped in creating the mobile application, provided testing for various parts of the project, and coordinated user testing. Igor Warszawski planned and implemented the GNN model used on the server. He also tested and validated the results, together with Filip Kozłowski.

Chapter 2

Biometrics in mobile devices - theory

Fundamental to the goal of the project was the use of biometric data in user identification. Biometric data can be defined as measurements of some unique characteristics of an individual. These can largely be divided into two main categories: physiological data, which is the measurement of the inherent characteristics of an individual's body, such as a fingerprint, an iris scan or a face scan, and behavioral data, which measures the person's movements, behaviors, speech patterns etc. [1]

Uniqueness of one's body is well known in biology. Features that may be used for person's identification are for example (FIX SOURCE: https://www.biometricsinstitute.org/what-is-biometrics/types-of-biometrics/):

1. **DNA** – found in cells of the living organisms, this acid carries genetic information.
2. **Eye features** – human iris, retina and scleral veins can be used in eye scans.
3. **Face** – full face scan is often used for user recognition, for example in mobile devices and laptops. (FIX SOURCE: https://developers.google.com/ml-kit/vision/face-detection, https://support.apple.com/en-us/102381)
4. **Fingerprints and finger shape** – fingerprints are widely used in forensics (FIX SOURCE: https://www.nist.gov/forensic-biometrics) and in digital scanners on mobile devices and laptops.

Other, less popular ways of identifying a person are for example: ear shape, gait, hand shape, heartbeat, keystroke dynamics, signatures, vein scans and voice recognition.

One possible way to extract data from a person's behavior is via *keystroke dynamics*. This type of behavioral biometrics is acquired from a user by means of a keyboard or other typing device and records and extracts features from the way the keyboard is used. Most commonly used and almost universally applicable to any keyboard device is the measurement of timings between each character typed. If the user uses a physical keyboard, it is also convenient to derive the following features [6]:

1. **Hold Time** – time between key press and release
2. **Down-Down Time** – time between first key press and second key press
3. **Up-Up Time** – time between first key release and second key release
4. **Up-Down Time** – time between first key release and second key press

5. **Down-Up Time** – time between first key press and second key release.

It can therefore be said that keystroke dynamics focus mostly on identifying user's rhythmic patterns in their keystrokes. Such data can be used in conjunction with for example a password or a passphrase as a means of additional protection against password theft – this idea was already being tested in 1990 by Joyce and Gupta (FIX SOURCE). The concept itself was already being studied in 1980, with Gaines et al. (FIX SOURCE) experimenting on secretaries' typing latency data. By 1997, clustering methods were already being used in experiments on user data on a small scale (42 profiles) by Monrose et al. [4]. Algorithms used for such data evolved after that point and the raise in popularity of neural networks.

Lu et al. [3] used a combined CNN+RNN approach to obtain the results of ERR of 2.36% on Buffalo dataset and 5.97% on Clarkson II datasets. Çeker and Upadhyaya (FIX SOURCE) used a CNN to create a multiple classification model – this method is mostly suitable for smaller datasets with smaller number of users, as opposed to creating a personalized model for each user. This method had an ERR of 2.3%.

Another aspect of the keystroke dynamics recognition methods is how and when the data is collected. The systems can either work with some specific strings being typed by the user (like in Çeker and Upadhyaya case) or with the users being free to type anything within some length constraints (like in Lu 2020 [3]). In this project, the second approach was chosen as the more realistic one.

With some keyboards it may be more difficult to gather all the possible features. Even basic feature, such as the hold time can prove difficult to gather when using for example GBoard on mobile devices, which does not naturally send key press and key release information to the application (SOURCE). This information can thus only be gathered in approximation or by building another virtual keyboard application. This, however, has its drawbacks. The users are generally used to one type of keyboard (on mobile it may be for example GBoard or SwiftKey), so forcing them to use another type of keyboard may be detrimental. Same person may write somewhat differently on different keyboards and machines. This study includes a small subsection on cross-smartphone compatibility of the model, for example concerning two users using each others' smartphones.

While the model may be less accurate because of the lack of features, there can be some ways to mitigate it. Some other features can be added, which are largely specific to mobile devices, such as accelerometer data, or a larger sample can be used. A few of those options were considered by the researchers, and the results will be discussed in the next chapters (???).

The keystroke identification can also rely on other data gathered from the keyboard, such as the specifics of letters used, their average frequencies, most common connections between the letter or other statistics [7]. These statistics can be modeled in many ways. If the average Up-Up Time between two keys is gathered from the data, a graph can be formed, having additional features as see fit by the designers. Such graphs were constructed for the Neural Network models constructed in this study, which will be discussed in the next chapter.

According to EU guidelines, all data used in Machine Learning models should also be ethically sourced (FIX SOURCE: EU GUIDELINES `https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A32024R1689`). In this study, all data was sourced from willing participants and anonymized using unique ID numbers given to the participants by the researchers.

Chapter 3

Graph Convolutional Networks - theory

Graphs can defined as mathematical structures G consisting of a set of vertices V, a set of edges E and an incidence function ϕ, along with many variations and generalizations to such structure, can be used for describing entities, which are related to each other in some way. An example of such model could be a computer network graph or citation network. Neurons can also be modelled in a similar way. Relation data can often be best described using such graphs. [2]

Some problems relating to such data can be solved using Convolutional Neural Networks – this can also be the case for keystroke dynamics data, such as with Lu et al. [3] or Sharma et al. [6]. However, it can be reasoned that the Graph Neural Networks can also perform such tasks, with connections in graph data being used more directly in the model itself.

3.1 Graph Neural Networks

Graph Neural Networks (GNN) are networks designed for graph inputs. The resulting outputs are also graphs (specifically, they are node embeddings representing a graph), allowing for transforming information in the graph's nodes, edges and global context, such as metadata about the graph, aggregated information, graph features etc. [5]. GNN do not change the connectivity of the input in the output. ...

3.2 Convolutional Networks and Graph Convolutional Networks

TODO: how do GCNs and GCN networks differ? Network design is better left for project model section.

3.3 Graph-level prediction in GNN

Supervised learning on graphs can be achieved by labeling either nodes, edges or whole graphs. In typical training pipeline, an input graph is transformed into a node representation accepted by the network, which then transforms the data in the manner described above. Output node embeddings are then used to create a prediction head, which is used, together with labels and some loss function and evaluation metrics, for the prediction task. There are different prediction heads for node-level, edge-level or graph-level prediction [2]. For nodes, predictions can be made directly using node embeddings – this can be done by using a classification layer, like a dense layer [5]. For the edges, this must be done on pairs of nodes. For global graph predictions a pooling of node embeddings can be performed. Options for pooling include for example global mean pooling, global max pooling or global sum pooling [2].

Chapter 4

Gathering keystroke data on mobile devices

There are many ways to recognise a phone user using biometrics, such as scanning fingerprints or facial recognition. It is very useful for security purposes. The ease of use and reliability have made passwords less popular and led to their replacement by biometrics. However, since other biometric methods are also available, it is reasonable to test if biometrics derived from writing button press intervals and phone orientation could also be a reliable way to recognise the user. To collect data and test the results, the mobile application was created. The main goal of the application is to gather data with an easy-to-use, intuitive interface, send the data to a server for training purposes, check if the model recognises the user.

As previously stated, State of the Art models can actually perform well (FIXSOURCE) on such data. These models are however usually trained on data gathered from physical keyboards. Additionally, the Neural Network model created for user identification was chosen to be based on Graph Convolutional Networks, which differ from models used by many researchers in the past (FIXSOURCE). Because of that, an important part of the project was a study of results and data gathered, which is presented in chapter 3.4 and 3.5.

TODO: find statistics and add them to sources

4.1 Use cases

TODO: add use cases and a short paragraph explaining reasoning behind the project.

4.2 Server structure and communication with the application

TODO for FK: technical docs for the server and connections between the app and the server. Data layer can also be touched a little.

Mobile application can communicate with the server, which can be locally hosted on a computer. The programmer needs to...

Server uses FastAPI, which is...

Server has the following endpoints, which are used by application for...

4.3 Mobile application for data gathering and model testing

The application was written for Android devices supporting Android 8.1 or newer. As of 2024*, more than 93% of Android devices should be compatible. The Android platform was chosen, as

it was easier to test on and find a study group of the Android users as opposed to the iOS users (according to * , significantly more people in Poland, where the researchers are based in, use Android devices).

Technology used in the mobile application itself was Jetpack Compose, which is quoted by Google to be "Android's recommended modern toolkit for building native UI" (from site*). Language used was Kotlin. Persisence was achieved by using Android Room, which provided an abstraction layer over SQLite database, which was used for data collection.

TODO for KM: add statistics sources (Internet), some technicals about the inner workings of the app. How the data is stored, what is gathered and when. If you have any doubts, feel free to ask about any methods/composables. I will be updating method descriptions/docs soon – JG
TODO: fix from MVC to MVVM (this is the structure that JC uses).

4.3.1 Model View Controller and DataStore

The application uses Model-View-Controller (MVC) design pattern to support a clear separation of concerns.

- **Model:** Data is modeled using `KeyPressEntity` class, which represents a single key press event. It includes:
  - **Key** (`String`): The key pressed by the user.
  - **Press Time** (`Long`): The exact timestamp of the key press event.
  - **Duration** (`Long`): The time elapsed since the last key press event.
  - **Accelerometer Data** (`Float`): Currently unused but useful for future developing of the project.

```
data class KeyPressEntity(
    @PrimaryKey(autoGenerate = true) val id: Int = 0,
    val key: String,
    val pressTime: Long,
    val duration: Long,
    val accelX: Float,  // Accelerometer X axis
    val accelY: Float,  // Accelerometer Y axis
    val accelZ: Float,  // Accelerometer Z axis
)
```

  The `KeyPressEntity` is stored in a local SQLite database via Room.

- **View:** The user intefrace is implemented using **Jetpack Compose**, a declarative UI framework. Key components of the view contain:
  - **Input Fields:** Lets users enter their credentials (University ID) and use the application for training or testing by pressing keys.
  - **Completion progress:** Informs users on what phase they are and displays progress of completion, linked to the `phasesCompleted` state in the `MainViewModel`.
  - **Buttons:** Used for logging in, logging out, jumping phases and sending or downloading the data collected through training or testing stage.

- **Controller:** The role of the controller is fulfilled by `MainViewModel`, which manages the application logic, handles interactions between the model and the view, and maintains the

state of the app.
The `MainViewModel` class manages this operations through:

- **Logic Handling:** Methods such as `login()`, `logout()`, `clearDatabase()`, and `onKeyPress()` are responsible for managing user state and data.
- **State Management:** Stores states `isLoggedIn`, `username`, and `phasesCompleted`, which are used to dynamically update the user interface.
- **Data Management:** Connects with the `keyPressDao` database to process data. `onKeyPress` saves key press events into the database, `exportDataToTsv` exports the collected data into TSV files.

In the app **DataStore** is used for storing login state and the user's ID. It has been implemented in `UserPreferences` class, and stores data such as:

- `LOGGED_IN_KEY` - login state
- `USERNAME_KEY` - user's ID.

This data is stored in the app's preferences file and can be accessed via dataStore object using:

- `isLoggedIn` - returns login state as `Flow<Boolean>`
- `username` - returns user's ID as `Flow<String>`
- `setLoggedIn()` - saves login state and user's ID into DataStore

The use of DataStore enabled the data to be stored securely, accessed and modified easily, and it is always available, which makes it a reliable and efficient way to manage user preferences and app state.

Model View Controller and DataStore...
Data was modeled as...
Data was saved...
Application design...
Training screen...
Testing screen...
Communications with the server...
Data sent to the server and downloaded locally...
Those should be subsections

Chapter 5

GCN Model

5.1 Choosing features for Neural Network model

5.1.1 Graph creation and feature encoding

TODO write properly not bulet points
The input for graph creation consists of two main parts, the duration of time between individual key presses, and the character of the key that was pressed. A natural way to represent such input was to map each unique character in the input sequence to a node in the graph. Dierected edges were added between nodes that represent charactes apearing after each other in the input sequence. TODO: add example graph visualization here Each time the same pair of characters appears in the input text it maps to the same egde. For each such pair, the duration is added to a list atributes for that egde, which will be aggreagted in later stages, to a form suitable for the GCN model. It would also be possible to model such pairs using a multiegde graph, as such models have been shown to perform well in other domains TODO zacytuj "multi-edge graph for convolutional networks for power systems. However we did not consider this approach.

Citation needed - some RNN paper that sequences of keys are important
End of TODO
Having chosen this graph structure, there are many ways to encode the input data in a form suitable to an GCN. Firstly, the edge atributes, need to be converted into node features. We found two ways to encode aggregate this information into node features.
For each node i:

1. Two values representing the average duration before and after the key represented by i was pressed.

2. Add two-dimentional vector of values, of size [number of allowed characters, 2]. Each key that can be found in the input is assinged a number. The n'th row in the vector corresponds to a node, with a key assined the number n, now called node j. The n'th row contains two values: the average duration on the edge from i to j and the average duration on the egde from j to i. The values for which egdes do not exist were assigned 0.

The clear difference between these two approaches is the level of aggregation. Method 1 aggregates all the egde information into 2 values, while method 2 aggregates it into a vector of values, although it imposes some limitations, such as asigning each key a unique index into this input vector. Furthermore method two increases the overall size of the input data and complexity of the model.

Similarly, there is more than one way to encode key information into node features.
We considered several method:

1. One hot encoding of each key
2. One hot encoding + some symbols map to one value, numbers to one value
3. One hot encoding vector of uncased letters without diacritical marks, special keys, one value for symbols, one value for numbers. Two additional bits, one for capitalized letters, one for letters with diacritical marks.

Citation needed: That paper that said node id's are nice. Again as before, these methods differ by degree of agreagtion. Methods 2 and 3 perform some sort of compression, mapping multiple characters to the same values, while method 1 provides a unique, one hot encoded identifier for each possible node. Cication found that such encoding helped the model to learn certain structures in the data. However, Slajdy z stanfordu ze tylko jak jest ograniczona liczba liter notes, that providing node identifiers as input features work well only for a small and known set of possible input nodes. While this requirement appears to hold true for this specific task, we found this is not the case. Many letters, which appear to be common do not apear in our dataset. TODO Which chars never appear. This means that the behaviour of the models would be unpredicitble for nodes with such identifiers. Furhter more, some charaters, for example EXMAPLEEEE like '(', apear only once, leading models to overfit and generalise poorly.

ADD histogram of how many times each character appears
ADD Which chars never appear

5.1.2 Feature selection - accelerometer data

5.2 Graph Convolutional Network for user recognition

In modern Machine Learning, a popular type of Neural Network is a Convolutional Neural Network. Such networks generally operate on grids. A Convolutional Neural Network (CNN) has a fixed node ordering – some input must firstly be mapped into a grid to be used with a CNN. There are ways to map many types of data into such format. Examples of researchers using CNNs for keystroke dynamics data include Sharma et al. [6] or Lu et al. [3]. In Lu et al. this involved applying the convolution layers over feature vectors, which were constructed in the following manner: for pairs of keys pressed in succesion in the sequence, a feature vector is created with fields: ID of first key, ID of second key, hold duration of first key, hold duration of second key, DD time (time between first press and second press) and DU time (time between first press and second release). After applying the CNN layer, GRU layers were used.

In this project, the goal was to use the graph networks that can naturally arise from keystroke data to – on a graph level – try to infere the users identity. The main difference was that the use of any keyboard already installed on the user's mobile phone causes some problems with gathering keystroke temporal data, as mentioned in the second chapter. Because of that, this project used mappings involving only key ...

TODO for IW: anything and everything about the model, the inner structure, the feature extraction can also go there, ask FK how you want to split these subjects up. FK will probably also check in with something here.

TODO: moreso about network structure, less about feature extraction

5.3 Model fine-tuning and hyperparameters - metrics

TODO for IW: write about the fine-tuning process, the metrics used and why are they used, cross-validations used etc. You can also post some hyperparam statistics here.

5.3.1 Metrics used

5.4 Testing model on users

TODO for xxx: when the tests are done (hopefully a week) we will discuss this. Also, there could be a part about many-users recognition.

5.4.1 Cross-smartphone user validation

TODO: what happens if two users train on smartphones that are not their own? What happens, if they cross-use their original model on another phone?

5.4.2 Discussion

TODO: discuss the findings.

Chapter 6

Conlusion

TODO for JG: not needed now, will write after the user tests are done.

Bibliography

[1] Hussien AbdelRaouf, Samia Allaoua Chelloug, Ammar Muthanna, Noura Semary, Khalid Amin, and Mina Ibrahim. Efficient convolutional neural network-based keystroke dynamics for boosting user authentication. *Sensors*, 23(10), 2023.

[2] Jure Leskovec and other instructors. Cs224w: Machine learning with graphs. Stanford University, Online Course Materials, 2024.

[3] Xiaofeng Lu, Shengfei Zhang, Pan Hui, and Pietro Lio. Continuous authentication by free-text keystroke based on cnn and rnn. *Computers & Security*, 96:101861, 2020.

[4] Fabian Monrose and Aviel Rubin. Authentication via keystroke dynamics. In *Proceedings of the 4th ACM Conference on Computer and Communications Security*, CCS '97, pages 48–56, New York, NY, USA, 1997. Association for Computing Machinery.

[5] Benjamin Sanchez-Lengeling, Emily Reif, Adam Pearce, and Alexander B. Wiltschko. A gentle introduction to graph neural networks. *Distill*, 2021. https://distill.pub/2021/gnn-intro.

[6] Atharva Sharma, Martin Jureček, and Mark Stamp. Keystroke dynamics for user identification, 2023.

[7] C. Wang, H. Tang, H. Y. Zhu, J. H. Zheng, and C. J. Jiang. Behavioral authentication for security and safety. *Security and Safety*, 3:2024003, 2024.