Capstone Project Phase B Behavioral Bio-metrics For Continuous User Authentication (22-1-R-2)

Dr. Dan Lemberg
Mrs. Elena Kramer

Rafael Elkoby - rafaelelkoby2@gmail.com
Dan Gutchin – dando400@gmail.com

June 2022

Contents

Abstract

Keystroke dynamics can be used to analyze how users type and interact by measuring various aspects of keyboard input gestures. Previous work demonstrated user authentication feasibility using keystroke dynamics. Our project considers various machine learning and deep learning techniques like CNN and RNN based on free-text keystroke features. Moreover, we will develop a simple UI to test new users. We will try to find the best authentication model for free text and mouse dynamics and compare it to related works in the field.

Keywords – 1D CNN, CNN, RNN, LSTM,GRU, Keyboard Dynamics, Authentication.

1 Introduction

Today, most of us have already used bio-metric authentication methods, including fingerprints and facial recognition, on our phones and laptops. Nevertheless, these methods mostly require dedicated hardware, which is not available across all devices like desktop PCs and basic laptop models. Another problem is that the methods in use today are primarily used for single-time authentication. Hence, these methods are used only in the login process, and once you pass one of the authentication methods, the system will require no further authentication from any user on a specific device.

Most computers (PCs and laptops) are not equipped with the hardware required for facial recognition and fingerprint scanning. Therefore, an alternative biometric authentication method that does not require any dedicated hardware will benefit the average user and the manufacturing company by eliminating the need to include these hardware additions in the product, which will lower the production costs for the company and hopefully for the company and the customer.

This project will present an approach that focuses on the most common hardware used today, the keyboard. Our approach is that by analyzing keystroke gestures, we can get a sufficient authentication of a user at the login process and during his use of the app/website.

The project document is arranged as follows. Section 2 discusses relevant background topics. Section 3 describes the technologies we used in our project. Section 4 will elaborate on the dataset we used and the data set we collected. Section 5 discusses the preprocessing we did on the data before starting the training. Section 6 will explain the models and their architectures. Lastly, Section 7 will describe the results of our research and will present what we achieved in comparison with other works.

2 Background and Related Work

2.1 Keystroke dynamics

According to [4], ”keystroke dynamics is not what you type, but how you type.” Most previous work on typing biometrics can be divided into either classification based on a fixed-text or authentication based on free text [5]. In 2009 Kevin S. Killourhy and Roy A. Maxion [2] published a keystroke dynamics benchmark dataset containing 51 subjects with 400 keystroke timing patterns collected for each subject. Besides, they evaluated 14 algorithms for keystroke dynamics on this dataset, including Neural Networks, KNNs, SVMs, etc. Various distance metrics were used, including Euclidean distance, Manhattan distance, and Mahalanobis distance. In the work of Han-Chih Chang, Jianwei Li, Ching-Seh Wu, Mark Stamp [1], they considered a wide variety of machine learning and deep learning techniques based on fixed text and keystroke derived features. For fixed text, they find that models based on extreme gradient boosting (96.4% accuracy) and multilayer perceptron (94.7% accuracy) perform well in their experiments. These works are mainly focused on fixed text. In the work of Xiaofeng Lu, Shengfei Zhang, they used the sliding window method and the combination of RNN and CNN 1.

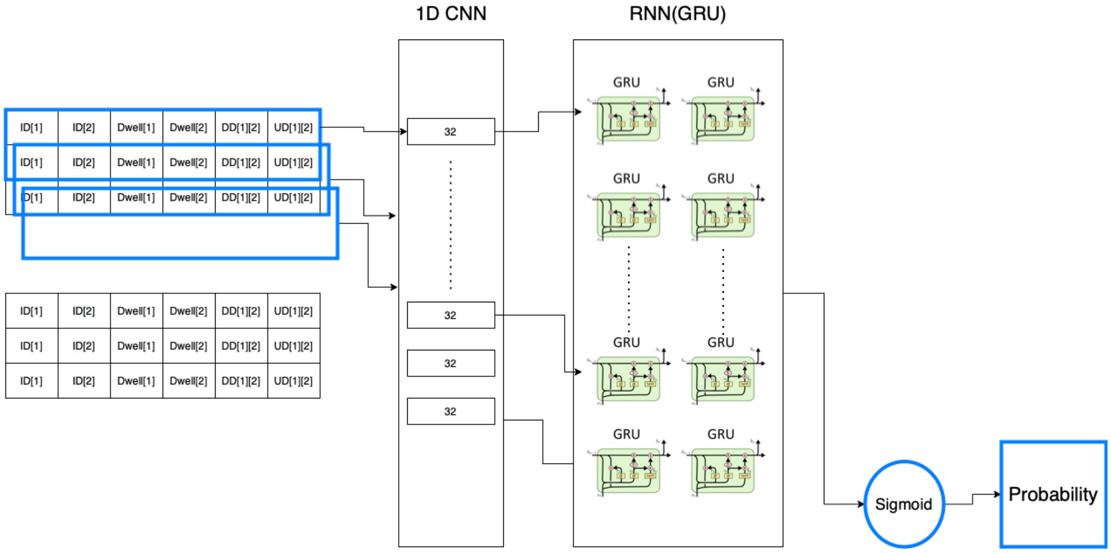


Figure 1: The approach presented in [3]

Different convolution kernels handle the keystroke sequence to extract the combination features from different positions. The blue box is a convolution kernel, the length of which is the width of the blue box. The convolution kernel slides over the keystroke sequence to produce a feature map. The pooled feature map is input into a double-layer GRU network. There are already works regarding both methods of supervised learning working separately and achieving relatively good results, as mentioned in [1] and [3]. Table 1 shows the results of RNN using LTSM on fixed text. Although these results are regarding different fields of user authentication using keyboard dynamics wherein [1], they are regarding a fixed text approach, and in [3], they refer to a free text approach.

Model	Parameters				Accuracy
	Input size	Hidden size	Num layers	learning rate	
LSTM	31	5	1	0.3	91.28%
BI-LSTM	31	5	1	0.3	90.02

Table 1: The results presented in [3]

There is an emerging problem when talking about more than one user authentication in both cases. In this case, we have two approaches as well. The first approach has separate networks for each user where the output is binary or using the same network for authenticating all users where the output is a specific user probability. When considering the unsupervised methods of several known algorithms and clusterization methods such as K means, The studies that researched the mentioned methods achieved relatively poor results compared with the supervised methods. In addition, they focus on fixed text while we want to focus on free text user authentication when considering keyboard dynamics. In the unsupervised methods, we encountered the same problem as in supervised methods, where we needed to research if one model for each user would be better than one model for all users. The desired outcome for the chosen method for keyboard dynamics is that it will work on small chunks of text and will provide a high accuracy authentication.

3 Research / Engineering Process

As we advance with our project, we encounter different keyboard dynamics challenges. Thus, our research is encountering different challenges in every new step. In the following segment, we will describe each one of them independently for you to understand the full picture at the end.

3.1 1D-CNN's

A CNN works well for identifying simple patterns within your data, which will then be used to form more complex patterns within higher layers. A 1D CNN is very effective when you expect to derive interesting features from shorter segments of the overall data set and where the location of the feature within the segment is not of high relevance. This applies well to the analysis of time sequences. Another application is NLP (Natural Language Processing), although there are better models for that today. The key difference is the dimensions of the input data and how the feature detector (or filter) iterates across the data. As described in figure 2, the kernel of a 1D CNN Moves in a vertical way while the 2D CNN iterates over 3D input not only vertically but horizontally as well.

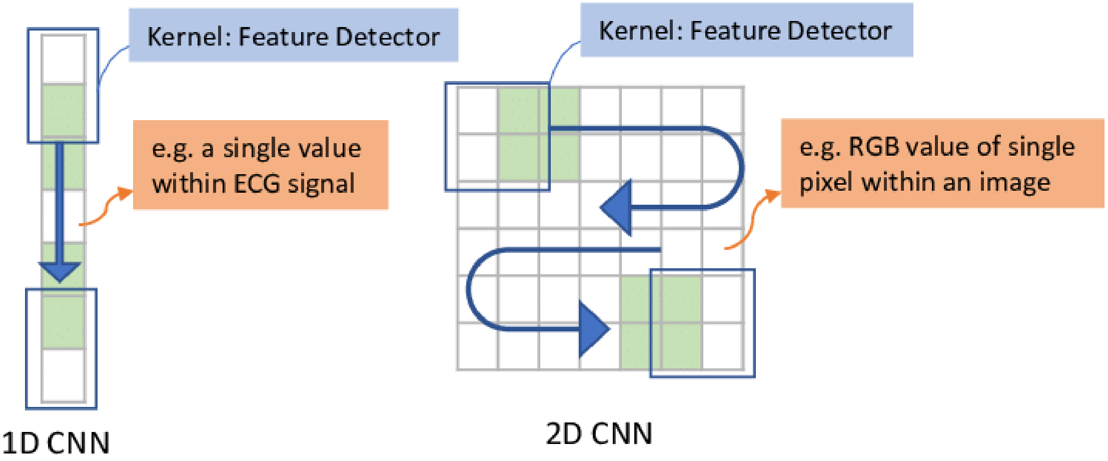

Kernel: Feature Detector
e.g. a single value within ECG signal
1D CNN
Kernel: Feature Detector
e.g. RGB value of single pixel within an image
2D CNN

Figure 2: The kernel movement on 1D CNN and 2D CNN

3.2 RNN's

RNN works on the principle of saving the output of a particular layer and feeding this back to the input in order to predict the output of the layer. An RNN can handle sequential data, accepting the current input data and previously received inputs. RNNs can memorize previous inputs due to their internal memory mechanism. RNN was created because there were a few issues in the feed-forward neural network, like handling sequential data, considering only the current input, and memorizing previous inputs. There are two main different RNN architectures like LSTMs and GRUs. LSTMs and GRUs were created as the solution to short-term memory. They have internal mechanisms called gates that can regulate the flow of information. These gates can learn which data in a sequence is important to keep or throw away. By doing that, it can pass relevant information down the long chain of sequences to make predictions. Almost all state-of-the-art results based on recurrent neural networks are achieved with these two networks. LSTMs and GRUs can be found in speech recognition, speech synthesis, and text generation. You can even use them to generate captions for videos.

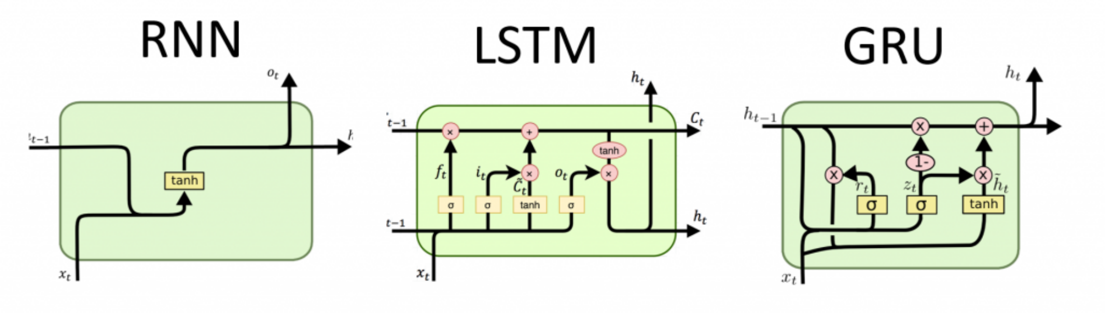


Figure 3: The LSTM, GRU and Simple RNN kernels

3.3 LSTM's

The core concept of LSTMs is the cell state and its various gates. The cell state act as a transport highway that transfers relative information all the way down the sequence chain. This is the "memory" of the network. The cell state, in theory, can carry relevant information throughout the processing of the sequence. So even information from the earlier time steps can make its way to later time steps, reducing the effects of short-term memory. The gates are different neural networks that decide which information is allowed in the cell state. The gates can learn what information is relevant to keep or forget during training.

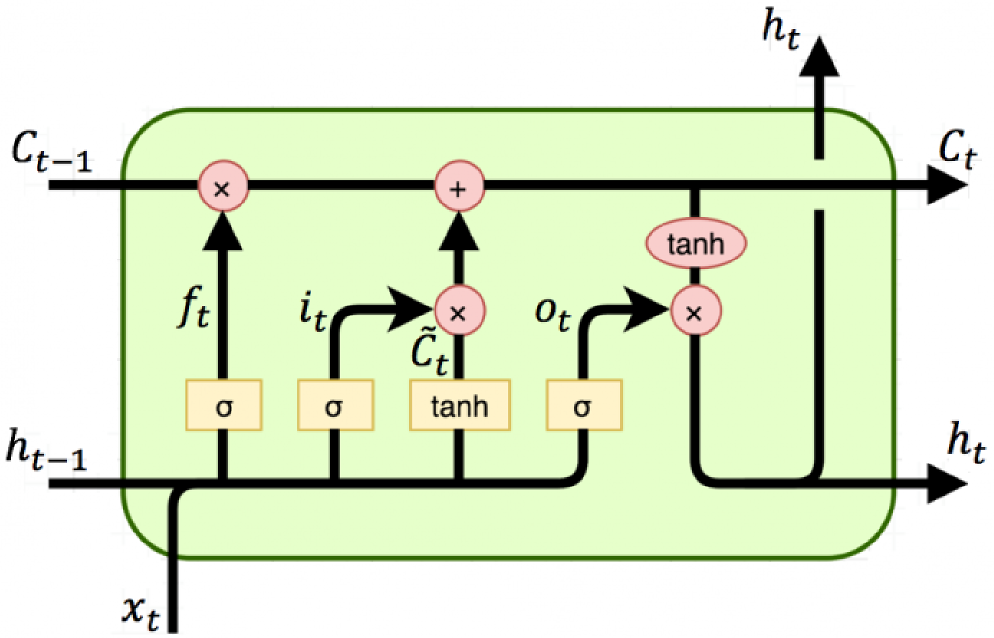


Figure 4: The LSTM kernel

3.3.1 Forget gate - f_t

This gate decides what information should be thrown away or kept. Information from the previous hidden state and information from the current input $x_t + h_{t-1}$ is passed through the σ function. Values come out between 0 and 1. The closer to 0 means to forget, and the closer to 1 means to keep.

3.3.2 Cell State - $\tilde{C}_t$

First, we pass the previous hidden state and current input into a σ function. That decides which values will be updated by transforming the values to be between 0 and 1. 0 means not important, and 1 means important. You also pass the hidden state and current input $x_t + h_{t-1}$ into the tanh function to help regulate the network. Then the output of the tanh will be multiplied by the i_t output. The result of the multiplication $i_t \times \tilde{C}_t$ will leave only the relevant data from the tanh and will give us the output $\tilde{C}_t$ which is the candidate data.

3.3.3 Output Gate

The cell state gets point wise multiplied by the forget vector $f_t \times c_{t-1}$. This has a possibility of dropping values in the cell state if it gets multiplied by values near 0. Then we take the output from the input gate i_t and do a point wise addition which updates the cell state to new values that the neural network finds relevant. That gives us our new cell state $C_t = i_t + (f_t \times c_{t-1})$.

3.4 Product

The final products of our project will be three different models and result in a wide variety of tests that will find the best performing model for the classification task. When we talk about free text feature extraction, we will use the most common features used today, And other keyboard dynamic works such as [4] [2] [3], where they are three main features that we will focus on.

1. **Hold time (dwell time)**
   The duration for which the key is held down means the time between pressing the button and releasing the button.

2. **Down-Down time**
   This is a time between when key1 was pressed, and key2 was pressed.

3. **Up-Down Time**
   This is the time between when key1 was released to when key2 was pressed.

An illustration for these features can be seen in figure 5.

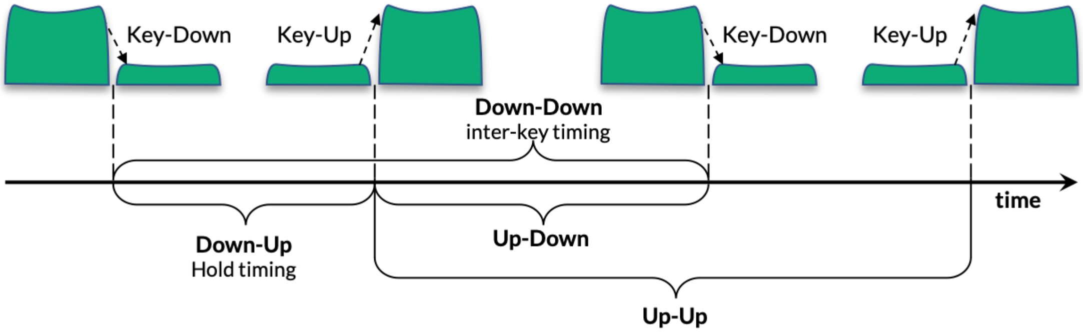


Figure 5: The keystroke timing scheme

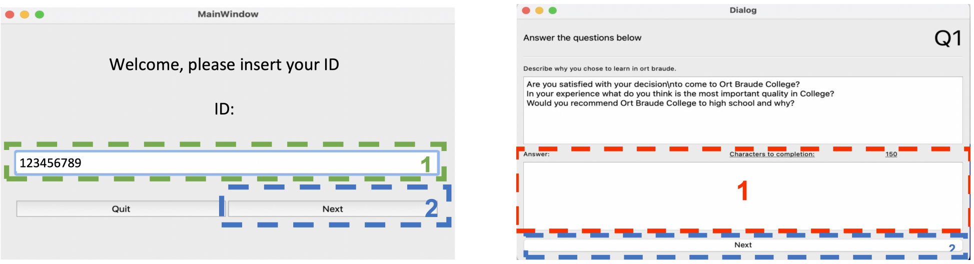
In addition to these features, we will experiment with adding the ASCII code of the pressed buttons. The reprocessing of the feature extracted will be described in a different section.

4 Dataset

We tried to collect the data by ourselves. As we proceeded with the project, we realized that the data collection process would take a long time and would require much more participants than we had. For that reason, we used the Buffalo keystroke dataset [6]. In the following sections, we will explain the data collection process we attempted and the Buffalo keystroke dataset.

4.1 Dataset collection

As a part of the project, we collected keystroke data from students using a program we built. The program was connected to an E3 amazon database server and used the MySQL engine for query interactions.
The first window presented gave the user a field to fill his ID number where a validation check then took place, and if the id was valid, the user was presented with the next window where the keystrokes were collected.



MainWindow

Welcome, please insert your ID

ID:

123456789

Dialog

Answer the questions below

Q1

Describe why you chose to learn in ort braude.

Are you satisfied with your decision\nto come to Ort Braude College?
In your experience what do you think is the most important quality in College?
Would you recommend Ort Braude College to high school and why?

Answer:

Characters to completion: 150

(a) Welcome window

(b) Question window

Figure 6

At the question window, the user is presented with a question and a text box where the typing takes place. The questions were selected randomly but were targeted to give us an answer that will sum to about 150 keystrokes. In the background, a keyboard listener is running and picking up keystrokes and time stamps from the user that is typing the answer. After completing all the questions, the parameters are calculated for every two adjacent keystrokes. The parameters are dwell time, up-down, and down-down-time. Then they are vectorized and sent to the database. In the database, the scheme is very simple, as described in table 2.

ID	Vectorized data
123456	[(0.23,0.1,0.032,0.3,0.23,0.03,0.09)......)]
851358	[(0.12,0.1,0.0612,0.26,0.34,0.02,0.019)...)]

Table 2: The scheme of the database

4.2 Buffalo Dataset

This dataset is collected from 148 subjects in 3 separate laboratory sessions; each session takes about 50 minutes and contains about 5.7k keystrokes. There are 28 days in average time intervals between sessions. Four different types of keyboards are used across sessions. There are two subsets of users divided based on the keyboards they use: baseline subset (ID from 001 to 075), with 75 users using the same type of keyboard across three sessions; and rotation subset (ID from 075 to 148), with 73 users using three different types of keyboard across three sessions.

5 Preprocessing

After we obtained the data of the buffalo dataset, we needed to preprocess the data so it would be ready to be fed to the CNN.

5.1 Conversion script

The data that was given in the Buffalo keystroke dataset [6] was collected in a different way than we collected it. The data have been collected into a .txt file, but the layout was different. The data was arranged as the following -

```
A KeyDown 63578429792961
A KeyUp 63578429793054
...
...
...
M KeyDown 63578429793257
M KeyUp 63578429793382
```

Where the first character represents the key pressed, then what action took place, and at the end, the timestamp. To overcome this problem, we created a conversion script to convert it to our format, which is as the following -

```
(ID[1] | ID[2] | Dwell[1] | Dwell[2] | UD[1][2] | DD[1][2])
```

The ID represents the key code of the pressed or released key, and the Dwell, Up-Down, and Down-Down times are calculated from the timestamps. To get the key code of every key, we created a map based on Microsoft key codes for Windows. In this article, we assume that the user is using a windows machine for the correction of the conversion script. At the end of the process, we had a file that contained a list of tuples of 6 as presented above.

5.2 Sliding window

Another preprocessing stage is the Sliding window. In order to obtain more data, we iterated over it and created blocks that vary in size from 20 to 50 tuples from every position, which we call a window; later on, we test the best performing block size for the classification task is. Afterward, we append all the windows and label them with the corresponding label, where 1 is the label that belongs to the user we want the model to authenticate and 0 if it is anyone else

(any other user from the dataset). This process happens for every session file from the Buffalo dataset.
To have a good ratio between the data of the person we want to authenticate and the other users, we take a portion from every file to get a 1:1 ratio between the data of the user and everyone else. At the time the data is collected or converted, the key ID's values are between 0 - 254, so we normalize them by dividing them by 254. The final process is to label all the data and reshape the data so it will fit the model specifications.

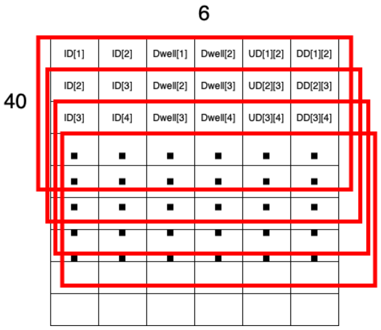


Figure 7: Sliding window visualization

6 Technologies

During the study, we used a number of technologies for writing code and training the model in the most effective manner we could.

6.1 Google Colab

Colaboratory, or "Colab" for short, is a product from Google Research. Colab allows anybody to write and execute arbitrary python code through the browser and is especially well suited to machine learning, data analysis, and education. Colab gives you access to expensive graphic cards that we will use to train the models. We bought the Colab pro plus, which gives you access to NVIDIA T4 or p100 and 52 GB of RAM.

6.2 HDF5

The Hierarchical Data Format version 5 (HDF5) is an open-source file format that supports large, complex, heterogeneous data. HDF5 uses a "file directory" like structure that allows you to organize data within the file in many different structured ways, as you might do with files on your computer. Because Colab uses google's drive as its system ROM, we needed to upload our data to the Colab framework. The problem is that the I/O in the Colab framework is slow, so a high-performance file architecture like HDF5 actually shortened the training time of our models.

6.3 LaTeX

LaTeX is a software system for document preparation. LaTeX is widely used in academia for the communication and publication of scientific documents in many fields, including mathematics, computer science, etc. Because of its popularity and simplicity, we decided to learn how to use LaTeX and wrote our article in LaTeX.

7 Models

In this section, we will describe three different types of architectures that we tried in our research. The Data we will test it on is from the Buffalo dataset, and we will be using a 10% portion for validation.

7.1 1D CNN + 1LSTM

The first architecture is a one 1D Conv net followed by an LSTM with 32 units. The first 1D layer outputs 32 features that are fed into the LSTM; the output of the LSTM is flattened with a dropout of 0.5 and connected to a fully connected layer with two neurons that use the Softmax activation function for the output.

Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, 39, 32)	416
cu_dnnlstm (CuDNNLSTM)	(None, 39, 32)	8448
dropout (Dropout)	(None, 39, 32)	0
flatten (Flatten)	(None, 1248)	0
dense (Dense)	(None, 2)	2498

Table 3: 1D CNN + 1LSTM Model summery

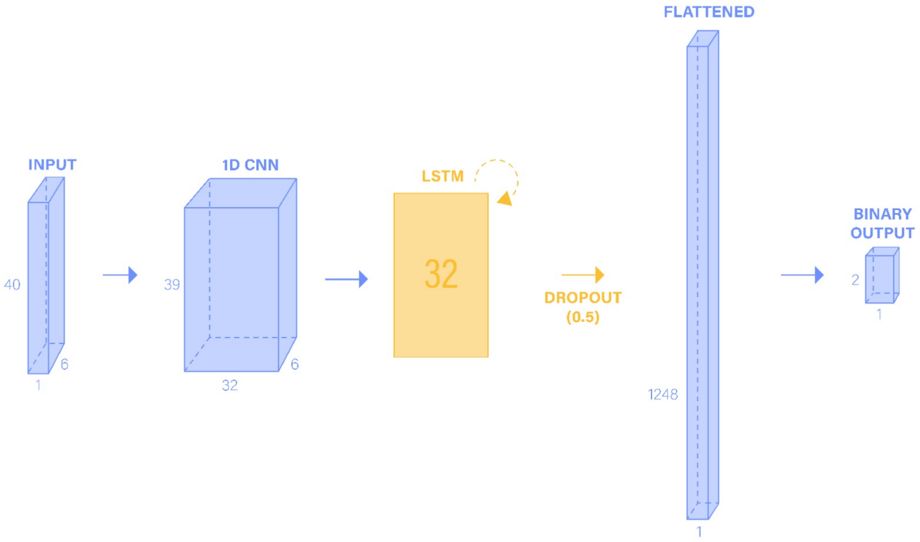


Figure 8: 1D CNN + 1LSTM visualization
with window size of 40

7.2 2LSTM

In the second architecture, we tried to abandon the first layer of the 1D convolution and replace it with an LSTM layer. Both layers have 32 units and a dropout of 0.5 after each layer. Then the output of the second LSTM is flattened and connected to a fully connected layer with two neurons that use a Softmax activation function

Layer (type)	Output Shape	Param #
cu_dnnlstm_2 (CuDNNLSTM)	(None, 40, 32)	5120
dropout_2 (Dropout)	(None, 40, 32)	0
cu_dnnlstm_3 (CuDNNLSTM)	(None, 40, 32)	8448
dropout_3 (Dropout)	(None, 40, 32)	0
flatten_1 (Flatten)	(None, 1280)	0
dense_1 (Dense)	(None, 2)	2562

Table 4: 2LSTM Model summery

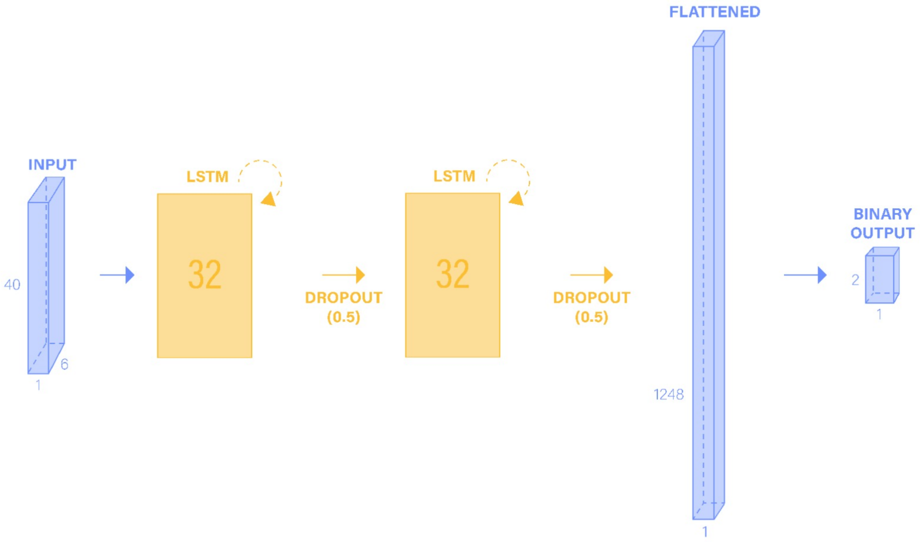


Figure 9: 2LSTM visualization with window size of 40

7.3 1D 2LSTM

The third architecture is a combination of the two models above. First, we have one 1D Conv layer with 32 features and a kernel size of 2, then 2 LSTM layers with 32 units and a dropout of 0.5 after each LSTM layer. This architecture obtained the best results, as we will describe in the next chapter.

Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, 39, 32)	416
cu_dnnlstm (CuDNNLSTM)	(None, 39, 32)	8448
dropout (Dropout)	(None, 39, 32)	0
cu_dnnlstm_1 (CuDNNLSTM)	(None, 39, 32)	8448
dropout_1 (Dropout)	(None, 39, 32)	0
flatten (Flatten)	(None, 1248)	0
dense (Dense)	(None, 2)	2498

Table 5: 1D CNN + 2LSTM Model summery

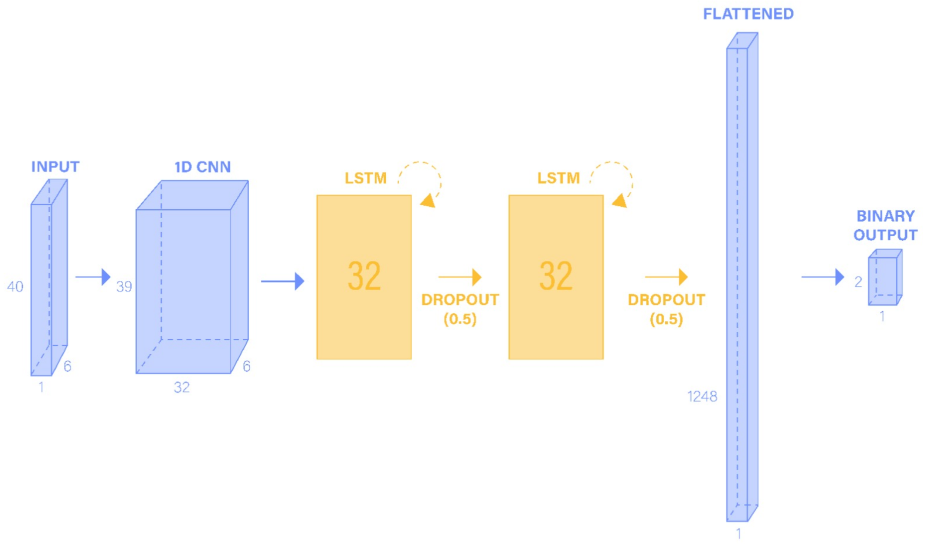


Figure 10: 1D 2LSTM visualization
with window size of 40

7.3.1 Dan 1D 2LSTM

This model is the same as the model presented above. The only modification to this model is that the model was trained using the LSTM layer instead of the cu_dnnlstm layer, which supports the Cuda core operations on Nvidia's GPUs. This change allows us to run the model after training on every computer without any dependency on GPU. The shapes are smaller on this model as our data is smaller due to time limitations.

Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, 39, 32)	416
lstm (LSTM)	(None, 29, 32)	8320
dropout (Dropout)	(None, 29, 32)	0
lstm (LSTM)	(None, 29, 32)	8320
dropout_1 (Dropout)	(None, 29, 32)	0
flatten (Flatten)	(None, 928)	0
dense (Dense)	(None, 2)	1858

Table 6: Dan 1D CNN + 2LSTM Model summery

8 Results

In this section, we will show the scores of all the models in a variety of validation Methods. All models were trained for 150 epochs. We used the Adam optimizer. The learning rate started from $1 \times e - 04$.After some observations with changing epoch numbers which vary between 50 - 1000 epochs, we found that there is usually a saturation that starts at epoch 100. As a result, we used learning decay, and we divided the initial learning rate at the 100^{th} epoch by 10 to $1 \times e - 05$. We tried window sizes of 50, 40,30, and 20. then we compared the results and chose the window size with the best accuracy and loss. To train our models, we used Google Colab.

Model	Parameters				
	Window Size	Loss	Acc	Validation loss	Validation acc
1D_1LSTM	20	0.2494	0.8960	0.2424	0.9033
	30	0.1410	0.9414	0.1423	0.9410
	40	0.0686	0.9751	0.0831	0.9697
	50	0.0502	0.9817	0.0521	0.9829
1D_2LSTM	20	0.2507	0.8927	0.2335	0.9001
	30	0.1092	0.9558	0.1043	0.9561
	40	0.0757	0.9718	0.0824	0.9707
	50	0.0211	0.9927	0.0188	0.9957
2LSTM	40	0.1911	0.9198	0.1815	0.9277
Dan 1D_2LSTM	30	0.0131	0.9952	0.0123	0.9951

Table 7: Models performance compared by window size

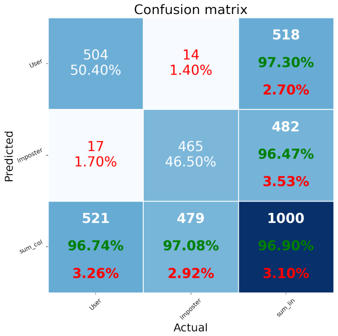


Figure 11: 1D_2XLSTM
WS = 40

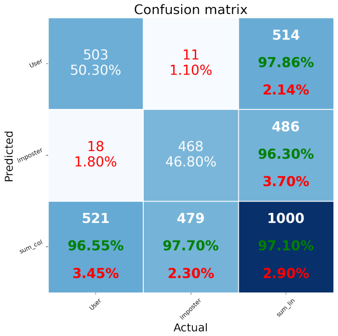


Figure 12: 1D_1XLSTM32
WS = 40

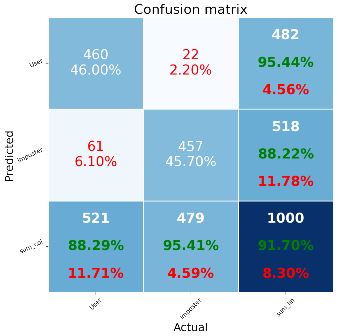


Figure 13: 2LSTM32
WS = 40

After training the models, we found that the 2LSTM model performs poorly relative to the other two models in a window size of 40, so we didn't proceed with that model. The 1D 1LSTM model and the 1D 2LSTM were a very close call. At table 7 we can see that even when the model accuracy is higher, the validation accuracy and loss are better at the 1D 2LSTM model. The reason for that is that the additional LSTM layer obtains finer features, and that will help the model to detect the user better and give better predictions when encountering new data. As we can see from the comparison results, the best choice for a continuous authentication window will be the window size of 30 using the 1D 2LSTM model, where the validation results are better and will give us a small enough window to have fast user authentication in real-time. Another parameter will be the F1 score, which will give us a score while taking into account the False Negative values(FN) and False Positive values(FP) that will help us to further base our decision to choose the 1D 2LSTM model. The F1 score will be presented in section 8.1. From a more macro perspective, we, of course, can understand that if real-time authentication is not needed, the better option will be the 50 window size, where the loss and accuracy are higher.

There is a crucial observation from table 7, we can see that the Dan 1D_2LSTM when using a window size of 30, has the best results. However, this result can be deceiving. When looking at the results, we can see that the results are better compared to the other models with a window size of 30, especially when looking at the loss and validation scores. This is a direct indicator that we lack data, leading to the personal classification inadequacy. The amount of data used to train this model using the data we collected is about 60% the size of the data used to train and validate the other models.

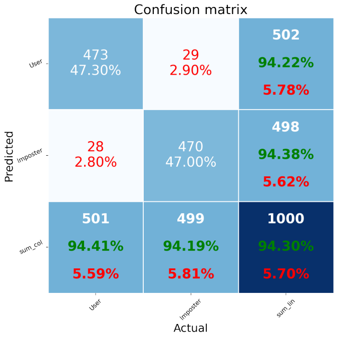


Figure 14: 2LSTM32
WS = 30

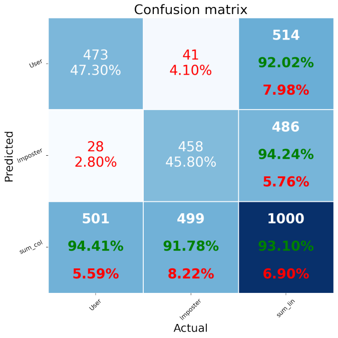


Figure 15: 1D_1XLSTM32
WS = 30

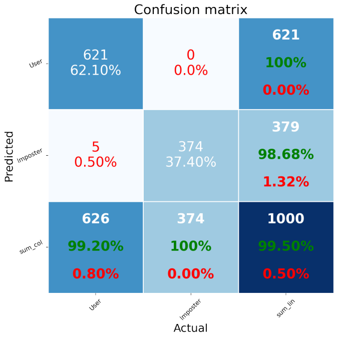
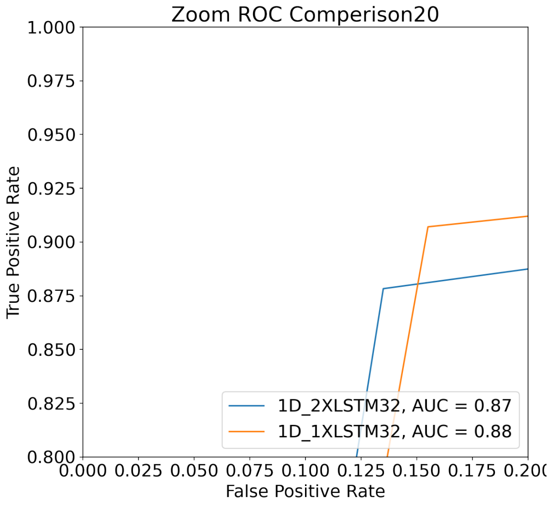
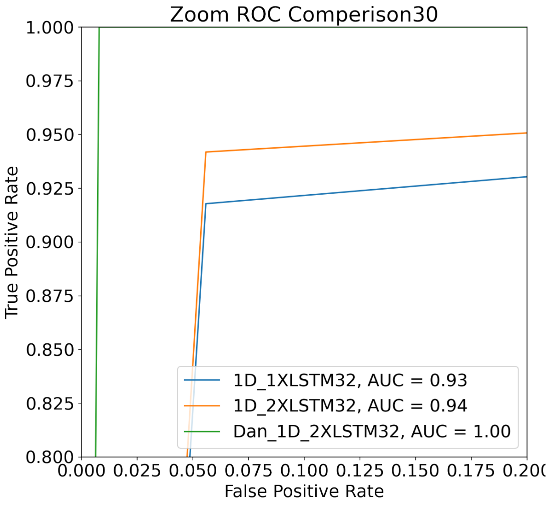


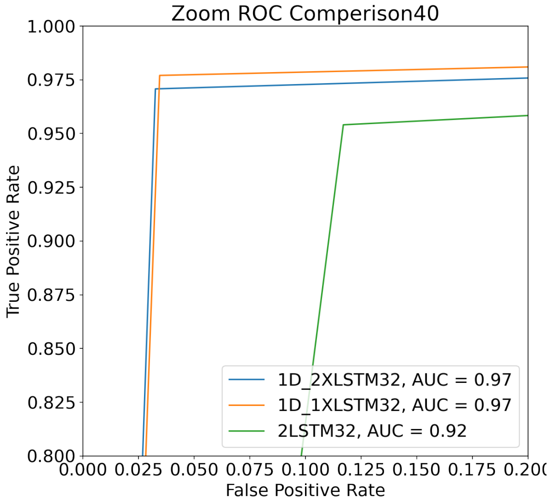
Figure 16: Dan 1D2LSTM32
WS = 40

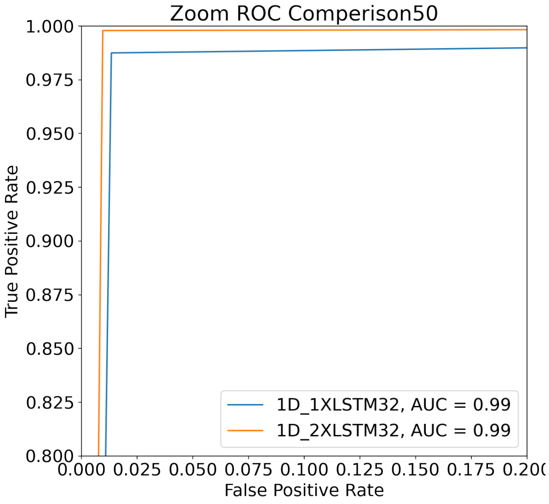
8.1 ROC AUC

A ROC curve plots TPR vs. FPR at different classification thresholds. Lowering the classification threshold classifies more items as positive, thus increasing both False Positives and True Positives. The following figure shows a typical ROC curve.









8.2 F1 SCORE

It is not possible to discuss the F1 score without first setting the context with precision and recall. precision and recall are metrics that help us evaluate the predictive performance of a classification model on a particular class of interest, also known as the positive class.
the questions we should ask ourselves when thinking about Precision and Recall are -
Precision: Of all positive predictions, how many are really positive?
Recall: Of all real positive cases, how many are predicted positive?

$$Precision = \frac{TP}{TP + FP} \tag{1}$$

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

Precision measures the extent of the error caused by False Positives (FPs), whereas recall measures the extent of the error caused by False Negatives (FNs). Therefore, to decide which metric to use, we should assess the relative impact of these two types of errors on our use case. Thus, the key question we should be asking is, ”Which type of error — FPs or FNs — is more undesirable for our use case?”.
In our use case, we assess that the errors caused by FPs and FNs are (almost) equally undesirable. Hence, we wish for a model to have as few FPs and FNs as possible. Put differently, we want to maximize both precision and recall. In practice, it is not possible to maximize both precision and recall at the same time because of the trade-off between precision and recall. The F1-score combines the precision and recall of a classifier into a single metric by taking their harmonic mean. We use it to compare our trained models, Where we see both precisions and recall to be equally impotent. In this case, the F1-scores for our models can be used to determine which one produces better results as we don't value the error caused by False Positives to be greater then the error caused by False Negatives.

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} = \frac{2 * TP}{2 * TP + FP + FN} \tag{3}$$

With that in mind, we can now understand that in our case, if an imposter will not be kicked out of the system, it will be as severe as if a user will be kicked out if he is not an imposter. That is why the F1 score suits our case perfectly, as we want to have a balanced system and not emphasize the FP over the FN and vice versa. Now we can understand even further that the choice to use the 1D 2LSTM model is a good decision as it is better than the 1D 1LSTM model F1(0.9299) and close enough to the result of window size of 40(0.9677).

The results for our models can be found in table 8.

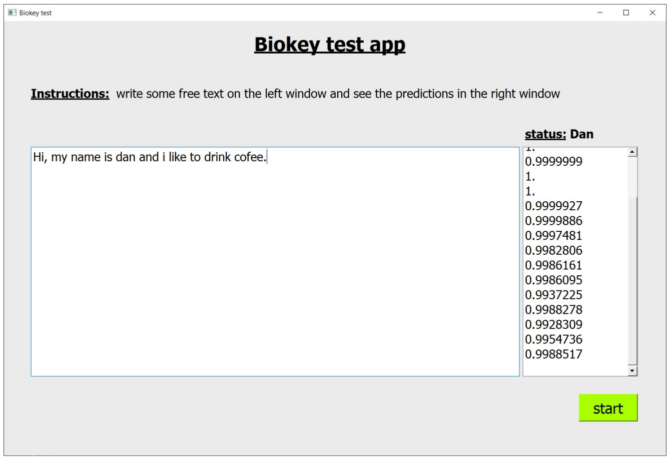
Model	Window Size	F1
1D_2XLSTM32	20	0.8602
	30	0.9428
	40	0.9677
	50	0.9937
1D_1XLSTM32	20	0.8658
	30	0.9299
	40	0.9699
	50	0.9864
2XLSTM32	40	0.9167
Dan 1D_2LSTM	30	0.9933

Table 8: F1 score across all models with several window sizes

9 System experiment

9.1 General

To test our models in real-life scenarios, we built a simple UI that simulates a continuous authentication environment where the user types free text and the system gives a predictions for every window. The model we used to provide prediction is the Dan_1D_2xLSTM32 model. more information about the model can be found under the "Models" section of this article. While typing, the test app will give a prediction for every window visible on the right of the text box. The UI is presented in figure 18



Biokey test app

Instructions: write some free text on the left window and see the predictions in the right window

status: Dan

Hi, my name is dan and i like to drink cofee.

```
1.
0.9999999
1.
1.
0.9999927
0.9999886
0.9997481
0.9982806
0.9986161
0.9986095
0.9937225
0.9988278
0.9928309
0.9954736
0.9988517
```

Figure 17: The system UI

9.2 Experiment results

When evaluating a biometric system, different terms are used to analyze the performance of biometric security systems. To get a good feeling of what the system is capable of, we first will explain the terms used to evaluate it and then show the results.

9.2.1 Terms

FAR and FRR

FAR and FRR. Anyone who wants to assess or compare the performance of biometric security systems cannot ignore these terms. This section explains what FAR and FRR mean.

- **False Acceptance Rate (FAR)** - The proportion of times a system grants access to an unauthorized person
- **False Rejection Rate (FRR)** - The proportion of times a biometric system fails to grant access to an authorized person.

It is essential to notice that as the number of false acceptances (FAR) goes down, the number of false rejections (FRR) will go up and vice versa. If you try to reduce the FAR to the lowest possible level, the FRR will likely rise sharply. In other words, the more secure your access control, the less convenient it will be, as the system falsely rejects users. The same also applies the other way round.

Equal error rate

Equal error rate (EER) is a biometric security system algorithm used to predetermines the threshold values for its false acceptance rate and its false rejection rate. When the rates are equal, the common value is referred to as the equal error rate. The value indicates that the proportion of false acceptances is equal to the proportion of false rejections. The lower the equal error rate value, the higher the accuracy of the biometric system.

9.2.2 Results

We conducted four sessions where 3 were intruders, and the other 1 was the user that the NN is trained to recognize. The participants are all the ages 25-27. all participants used the same keyboard in the same lab environment. When considering a threshold of 50% for the FAR and a 70% for the FRR, we got the results as in the table 9

FRR	FAR	EER
1.094%	19.104%	8.274%

Table 9: Test results in percentage

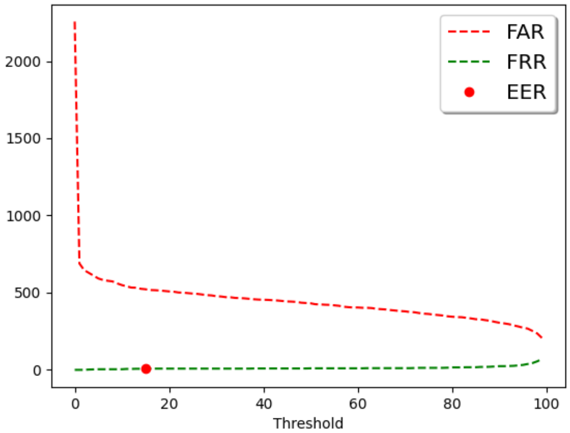


Figure 18: FFR and FAR as a function of the threshold in percentage

It is essential to mention that we lack data. With more data, better results can probably be achieved.

10 Conclusions

In this research, we sought to improve existing models regarding binary classification for free text input. We found that existing models can be improved. In our models, we saw an improvement by replacing the GRU module with an LSTM module with 32 units and even adding another LSTM layer. Moreover, we found that a window size of 30 preserves relatively good results compared with larger window sizes. In addition, as mentioned in [3], we encountered a similar problem of lack of data that leads to inadequacy of the personal classification.

References

[1] Han-Chih Chang, Jianwei Li, Ching-Seh Wu, and Mark Stamp. Machine learning and deep learning for fixed-text keystroke dynamics. *arXiv preprint arXiv:2107.00507*, 2021.

[2] Kevin S Killourhy and Roy A Maxion. Comparing anomaly-detection algorithms for keystroke dynamics. In *2009 IEEE/IFIP International Conference on Dependable Systems & Networks*, pages 125–134. IEEE, 2009.

[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*, pages 48–56, 1997.

[5] Nataasha Raul, Radha Shankarmani, and Padmaja Joshi. A comprehensive review of keystroke dynamics-based authentication mechanism. In *International Conference on Innovative Computing and Communications*, pages 149–162. Springer, 2020.

[6] Yan Sun, Hayreddin Ceker, and Shambhu Upadhyaya. Shared keystroke dataset for continuous authentication. In *2016 IEEE International Workshop on Information Forensics and Security (WIFS)*, pages 1–6. IEEE, 2016.