Keystroke Classification of Motion Sensor Data

An LSTM Approach

Marcus Patricks

Master's thesis
2021:E68

Lund University

Faculty of Engineering
Centre for Mathematical Sciences
Mathematical Statistics

Abstract

With mobile phones often being used to type sensitive information, it is important that they remain secure and leak no information. One conceivable channel for information leakage, though, is the motion sensors accelerometer and gyroscope, sensors that require no permission to be used by an app. Do the data they produce contain information about the keys being typed?

To answer this question, this thesis investigates whether, using LSTM networks, keystrokes can be classified as (1) either backspace or not backspace, and (2) any key on the keyboard, using only motion sensor data collected around the keystroke. Furthermore, the problems are investigated in three different cases, one where the models are built on a user-basis, one where they are built on a mobile phone brand-basis, and the last where they are built on a general basis, using data pertaining to all users and brands.

The thesis finds that the motion sensor data do indeed contain information about the keys being typed. The different cases yield similar results for the backspace problem (1), while models built on a user-basis performs best in the more general problem (2). Training on a user-basis yields an EER of 0.11 and an F1-score of 0.61 for the backspace problem, and an Accuracy of 51 % and a macro-averaged F1-score of 0.32 for the more general problem, much better than naïve model performance.

Acknowledgements

I would like to express my sincere gratitude to Johan Swärd and Daniel Maldonado at Callsign, without whom this thesis would be but a fraction of what it became. Their guidance, support and feedback have been a constant in the writing of this thesis. Furthermore, I would like to thank Andreas Jakobsson for introducing me to Johan and Callsign, and for his advice.

Contents

1 Introduction

1.1 Background

With the number of sensors in a normal mobile phone growing over the past decades, so does the diversity of the information they gather. Whereas early mobile phones only could receive and transmit phone calls, the modern smartphone lets its user capture videos and navigate using maps, giving it its ubiquitous role. With all this data being collected by sensors, some of it could certainly be misused, especially if collected without authorization. To deal with this, apps often need to request explicit permission to use many of the sensors, lest, for example, an app use the smartphone's camera without one's knowledge. Not all sensors need permission to be used by an app though, among these are motion sensors like the accelerometer and the gyroscope. Most smartphones use the accelerometer to count the number of steps the user takes, and the same could thus theoretically be done by any app installed. This information is perhaps unintrusive enough that even if it were to fall into nefarious hands it would pose no real threat to a user's privacy. Is it possible then to extract more private information from the accelerometer and gyroscope, or is the data they collect correctly deemed less private?

To investigate this, this thesis will explore the possibility of extracting keystroke data, what key is being struck on the smartphone's soft keyboard, given a stream of accelerometer and gyroscope measurements. For example, one could imagine typing a key on the left side of the keyboard, e.g. `"a"`, would cause the user to slightly tilt the phone to the left, thus perturbing the gyroscope measurements. If possible, such a stream would be highly private, as it could reveal for example the log-in credentials and private text messages of a user.

The thesis has been done in collaboration with *Callsign*, a company specializing in online security and authorization.

1.2 Aim

The general aim of extracting keystroke data is divided into two more specific problems:

Given the accelerometer and gyroscope measurements in close temporal proximity to a keystroke, classify it as

- either Backspace (1) or not Backspace (0) (henceforth known as the backspace problem).
- any standard character on the keyboard (the general problem).

Whereas the second problem is certainly a more interesting problem at first glance, the first problem is easier (and thus interesting as a step towards solving the second problem) and a concrete problem at Callsign.

While there surely are many different types of models suitable to solving the problems presented above, this thesis will focus on investigating Artificial Neural Networks' (ANNs) ability to solve them. Furthermore, both main problems will be tackled in three different cases each, namely:

- The **General** case (not to be confused with the general *problem*), where one large model is trained non-discriminatorily on data belonging to many users using different phone models.
- The **Brand-specific** case, where one model is trained for each mobile phone brand, with many users per brand.
- The **User-specific** case, where one model is trained for each user.

To investigate these cases and build the models we need a data-set that contains many examples done by many users on more than one phone model. Luckily, such a data-set exists and will be presented in the next section.

1.3 The Dataset

The *BB-MAS* data set [1] (Behavioral Biometrics Multi-device and multi-Activity data from Same users) is an open data set collected in 2017. It includes, among much else, accelerometer and gyroscope measurements collected while a user is writing text on the smartphone's soft keyboard, as well as information pertaining to the keystrokes themselves. The data was collected on two different smartphones, the Samsung Galaxy S6 and the HTC-One, the physical measurements of which one can find in Table 1. In total, 116 participants were tasked with writing both fixed sentences and free form answers to questions, with the phone in the upright portrait mode. The participants only used one of the phones each, no one used both the HTC and the Samsung models. In total this resulted in 544 935 keystrokes, or roughly 4700 keystrokes per participant. More information about and analysis of the data set can be found at [2].

Table 1: The physical measurements of the two phones used in the BB-MAS dataset. The dimensions are listed as Length x Width x Height [2].

	Samsung	HTC
Screen size (mm)	129.5	127
Resolution (pixels)	1440 x 2560	1080 x 1920
Weight (g)	138	160
Dimensions (mm)	143.4 x 70.5 x 6.8	146.4 x 70.6 x 9.4

The accelerometer data is structured in such a way that each sample contains the time of sample, along with the acceleration in all three device axes, see Figure 1. The gyroscope data, similarly, contains the time of sample and the rate of rotation around each axis. The sampling rate for the sensors was roughly 100 Hz, though at times the sampling is irregular and some timestamps contains multiple measurements. The keystroke data contains both the key-down and the key-up timestamps, describing when a key being pressed and released, respectively, and which key is pressed. The first four samples of the first participant's accelerometer and keystroke measurements can be seen in Tables 2 and 3. Gyroscope measurements are qualitatively similar to those of the accelerometer.

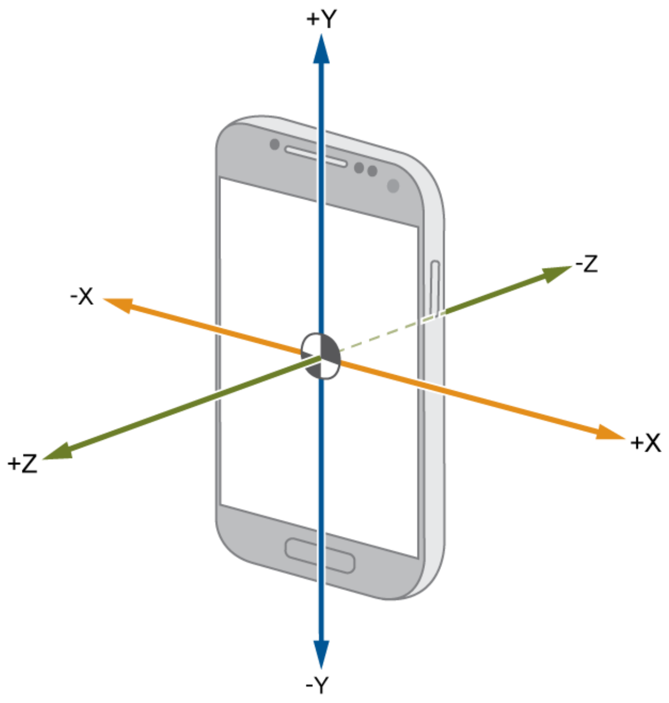


Figure 1: The axes of a smartphone [3].

Table 2: The first four accelerometer measurements for the first participant. Note how the magnitude of the acceleration is roughly equal to that which is exerted by gravity. EID stands for Event ID. Xvalue is the acceleration force along the x-axis, measured by the accelerometer, Yvalue and Zvalue are defined analogously.

EID	Xvalue	Yvalue	Zvalue	time
0	1.043872	3.245340	9.087193	2017-04-14 18:56:40.215
1	0.995988	3.303998	8.936357	2017-04-14 18:56:40.216
2	0.988805	3.355474	8.880095	2017-04-14 18:56:40.234
3	1.031901	3.456030	8.804677	2017-04-14 18:56:40.234

Table 3: The first four keystroke measurements for the first participant. A direction of 0 reflects a key being pressed, 1 a key being released.

EID	key	direction	time
0	t	0	2017-04-14 18:57:12.176
1	t	1	2017-04-14 18:57:12.352
2	h	0	2017-04-14 18:57:12.399
3	h	1	2017-04-14 18:57:12.517

1.4 Previous Work

A concise summary of somewhat recent research on the topic of side-channel (as opposed to direct, like an ordinary keylogger) attacks on mobile devices can be found at [4], with Section III being most relevant to the topics in this thesis. TouchLogger [5] and Taplogger [6] use hand-extracted features to infer PIN-codes from motion sensors. ACCessory [7] use other features to infer passwords on the QWERTY-keyboard, using Random Forests. Cai et al [8] investigated how the inference accuracy is affected by different devices, users and setting, finding that accuracy is lower if a user has not been seen by the model, but still substantially larger than that of randomly guessing. Wang [9] used ANNs on accelerometer, gyroscope and light sensor data to infer keystrokes with an accuracy of 48 % (over most letters of the alphabet).

2 Theory

Here follows a section describing the theory needed to understand the methods used in this thesis. The bulk of this section will revolve around intermediate topics regarding Artificial Neural Networks, ANNs, the basics of which will not be presented here.

2.1 RNN

While it is possible to use ordinary ANNs on sequential data, where one simply model each time-step (or equivalent) as an independent input node, it is possible to take advantage of the fact that the value at different time-steps are correlated with each other. One architecture that does this is the Recurrent Neural Network (RNN), the simplest of which can be seen in Figure 2.

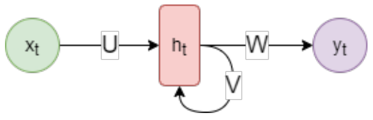


Figure 2: A simple RNN. W, V and U are weights and y_t, h_t and x_t are the output, hidden and input nodes, respectively, at time-step t.

The network is defined by the following equations:

$$\begin{cases} y_t = \phi_y(W \cdot h_t) \\ h_t = \phi_h(U \cdot x_t + V \cdot h_{t-1}), \end{cases} \tag{1}$$

where W, V and U are weight matrices (scalars in this case), ϕ_k activation functions and y_t, h_t and x_t are the output, hidden and input nodes, respectively, at time-step t.

The hidden layer does not just depend on the previous layer (in this case the input layer x), but also on the hidden layer in the previous state (where different subscripts symbolise different states). Some of the information stored in the last state can therefore travel to the next. To further understand the RNN, it is shown "unfolded" in Figure 3.

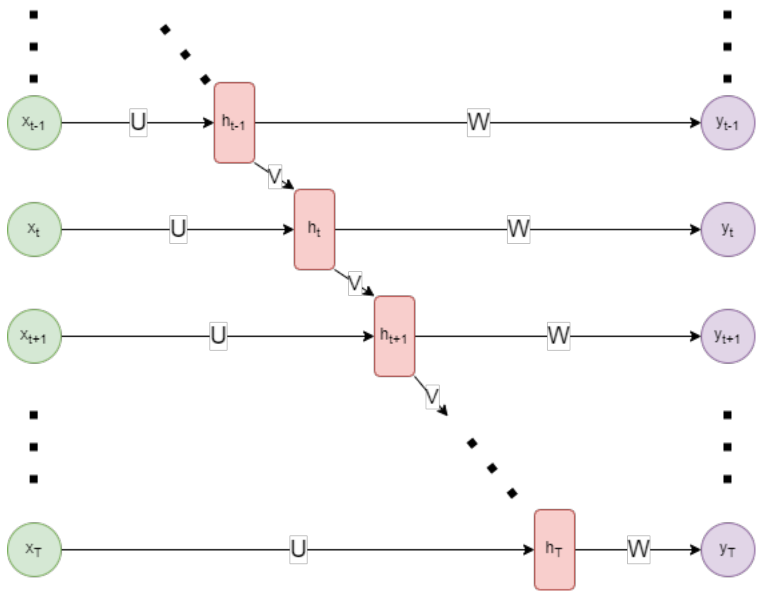


Figure 3: The RNN in Figure 2 unfolded.

While they may look different, the figures both describe the same network. Looking at Figure 3 one realises it is, in a sense, an ordinary ANN, though with many nodes and edges missing, and the weights between the hidden layers being equal by design. Also of note is that even the simple network in Figure 2 can be arbitrarily deep as the number of time-steps increases.

Back Propogation Through Time (BPTT) is probably the most used algorithm to train an RNN. It simply unfolds the network (transforms it from the representation in Figure 2 to the one in Figure 3), and then performs the same back propogation as in ordinary ANNs. Given the potentially immense depth of an RNN, though, the gradients used for the gradient descent have a tendency to either "explode" or vanish. This is due to the gradients being a product of T factors, where T is the number of time-steps in the input, and thus exponentially dependent on the size of the input [10].

2.1.1 LSTM

To solve the issue of vanishing and exploding gradients, Long Short Term Memory networks (LSTMs) were introduced in 1997 [11]. They share the "macro-structure" with ordinary RNNs, but the hidden nodes in the RNN are replaced with a more complex unit. A sketch of such a unit can be seen in Figure 4.

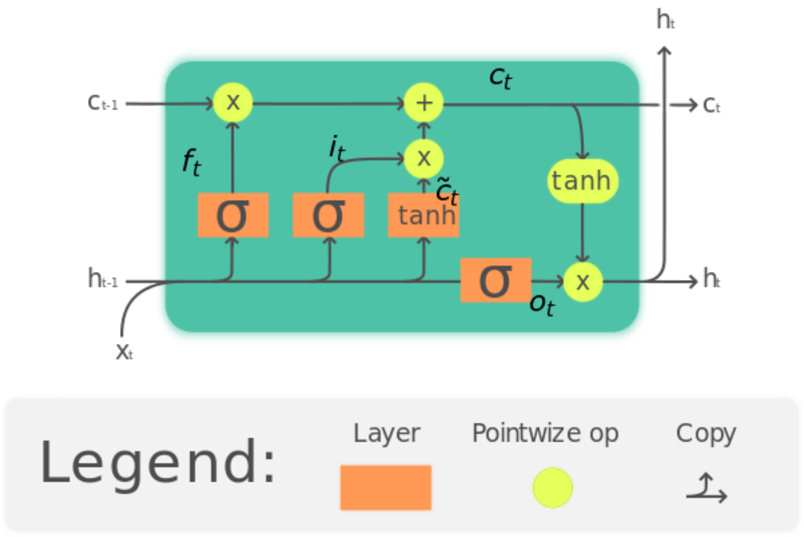


Figure 4: An LSTM Unit. Original by [12], edited by author.

The first thing to notice is that the LSTM unit has three inputs and two outputs, in contrast to a RNN unit which only has two inputs and one output. The new addition in the LSTM is the hidden "cell state" which is only passed to the temporally next LSTM unit (that is, the next state e.g. $t + 1$), not to the next layer, like h_t is in both LSTMs and regular RNNs. This lets a unit communicate with the next, without also communicating the same information to the next layer.

The LSTM unit consists of three "gates", in Figure 4 symbolized by an orange rectangle reading σ and the following pointwise multiplication. These regulate the "flow" of the signals c_{t-1}, c_t and $\tilde{c}_t$, determining how much should be let through. To understand the signals and gates, let's look at the functions defining the LSTM unit:

$$\begin{cases} f_t = \phi_g(W_f x_t + U_f h_{t-1}) \\ i_t = \phi_g(W_i x_t + U_i h_{t-1}) \\ o_t = \phi_g(W_o x_t + U_o h_{t-1}) \\ \tilde{c}_t = \phi_c(W_c x_t + U_c h_{t-1}) \\ c_t = f_t * c_{t-1} + i_t * \tilde{c}_t \\ h_t = \phi_h(c_t) * o_t \end{cases} \tag{2}$$

where W_k and U_k are weight matrices, ϕ_k are activation functions and $*$ symbolizes pointwise multiplication. The first three equations correspond to the three gates: the Forget, Input (or update) and Output gates. They each have independent weight matrices with biases (the biases being incorporated into the matrices to avoid cluttering). As suggested by Figure 4, the activation function ϕ_g is often sigmoidal. Next is the equation for the "candidate cell state" $\tilde{c_t}$, usually with the activation function $\phi_c = \tanh$. Then, the cell state, c_t and the output vector h_t.

The cell state of each cell is the weighted sum of the cell state of the previous cell state and the current candidate cell state, where the weights, not necessarily adding up to one, are the forget and the update gates, respectively. The candidate cell state $\tilde{c_t}$ can be seen as what the cell state *should be*, had the cell no memory of earlier states. The names of the two gates are aptly chosen, as the first one regulates how much should be remembered (or inversely, forgotten) from the previous cell, and the second how much should the cell state be updated with the new candidate cell state. The output gate in turn regulates how the output h_t depends on the cell state.

The LSTM unit presented above is only one of many possible variations, though probably the most common one, and a good starting point when dealing with sequential data. Greff et al. [13] tried nine different versions on three different problems and found that they all performed reasonably similarly on all problems.

2.1.2 Bidirectional LSTM

The reason for introducing the RNN was, as seen above, to preserve information from past time-steps, and one of the major reasons for introducing LSTMs was because they are simply better than regular RNNs at this. In many cases it may prove worthwhile to also use future information in a similar fashion. For this reason Bidirectional LSTMs (BLSTMs) were introduced. A BLSTM layer is constructed from two regular LSTM layers, where one processes data in the original, positive, direction, and the other in the negative direction. A sketch of a simple BLSTM network can be seen in Figure 5.

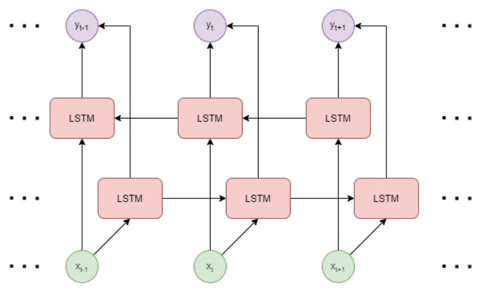


Figure 5: BLSTM network with inputs x_k, forward and backward LSTM layers and outputs y_k.

Bidirectional RNNs of any type was first shown to be effective when dealing with phoneme (unit of spoken sound) classification [14], as were BLSTMs [15]. Speech is often not understood "online", in the same instant as it is spoken; sounds are often understood only after a whole word is spoken, words after a whole sentence. The context of a word does not only depend on previous words, but also of the words coming next. It is difficult to say which types of data the context is only understood in conjunction with both the previous and the following data, where the use of BLSTMs is motivated. Other than phoneme classifications and other natural language processing problems [16][17], though, BLSTMs have been used with success in predicting wastewater flow rate [18], detection of kidney disease [19] and a number of other univariate time series classifications [20].

2.2 Transfer learning

Training large models can require a huge amount of data, and an equally large amounts of computational hours, making it unfeasible to create models from scratch for many problems. Many large models have already been trained, though, and perhaps it's possible to generalize some of their "intelligence"? This is indeed possible through a method known as Transfer Learning. The general idea is best exemplified by image classification using Convolutional Neural Networks (CNNs).

In deep CNNs the first couple of layers are often used to find general, low level structures, for example circle segments and edges. The latter layers then use this information to build higher level structures, used by the final layer to classify the image. Simply put, the latter layers ask "Are the circle segments and edges found by the previous layers located in such a way that implies that this image is of a car?" If instead of classifying images as "car/not car" one wants to classify different images as "firetruck/not firetruck", or indeed "dog/cat", the layers finding the low level structures could be reused, only specializing the latter layers for the task at hand.

Such methods have been used extensively, classifying widely different types of images. For example, VGGNet, introduced by K. Simonyan et al. [21], has been used for facial expression recognition [22], smoke detection [23], palmprint identification [24] and even epileptic EEG identification [25].

While Transfer Learning certainly seems to be effective when used in tandem with CNNs, the literature is less clear on whether it works well for LSTM networks. For example [26] shows that transfer learning can both improve or degrade model performance, depending on which data set is used in the base training and which is used in the fine tuning. One problem is that while it is fairly easy to see and understand what each layer of a CNN looks for and produces, it is much less obvious what an LSTM layer produces. Another is that most serial data lack a common "vocabulary", set of almost atomic units that build up the series, while most images have roughly the same vocabulary, like corners and edges. For transfer learning within corpora of similar vocabularies, like books in the English canon and Beatles lyrics, or gyroscope data collected from different users, the magnitude of this problem could reasonably be expected to lessen. Unsurprisingly, [26] indeed shows that transfer learning using two sets with a higher inter-data-set similarity performs better than two with low similarity.

2.3 Practical considerations

2.3.1 Batch training and padding

When training an ANN of any type, it is much more computationally efficient to train on many samples at the same time, in batches. This is done, in the case of sequential data, by constructing a 3D-tensor of size (BS, T, D), where BS is the batch size, T is the number of time-steps (or equivalent), and D is the dimensionality of the input data at a single time step. Thus, all samples need to be of the same length T, which is rarely the case. This can be remedied by organizing the data such that samples of the same length are placed in the same batch, or padding sequences with dummy values, often 0, until all sequences are of some chosen length T. All samples originally longer than T need to either be truncated or, more often, simply removed. When training, the model ignores the dummy values by not updating weights for time steps where the value is 0 for all dimensions. This is most easily done in TensorFlow using Masking layers, for more technical details see [27].

2.3.2 Class imbalance

A binary classification model classifying samples of a data-set in which the cardinality of the majority class is much higher than the minority, say 9:1, can easily reach an impressive accuracy of 90 % by always classifying as the majority class. This naive model is obviously useless, and the phenomena is known as the Accuracy paradox [28]. If the class imbalance is severe enough, an ANN will typically simply learn to neglect the minority class. Usually the resulting model is not as extreme as the naive model above, fully neglecting the minority class, but it often tends to such a behaviour. The problem is further exacerbated by the fact that the minority class is often of higher interest, and one would often be more tolerant of many false alarms than of a single missed positive. When building a model one thus needs to take class imbalance into account. As it may be difficult to improve the accuracy score, the first thing one needs to do is redefine what the goal of the model is, what metric to use as measurement of model performance. Precision, recall and their weighted harmonic mean, F-scores, may be suitable substitutions (the naive model above would have the worst F-score possible, 0). More information about precision, recall and F-scores will be presented in Section 4.1.1.

Equipped with a proper metric, now the model needs to train in a fashion that incentivizes classifying as the minority class. This is often done by either changing the loss function or the training data. In the first case, weights are assigned to each class, with high weights for uncommon classes. The loss of every single sample is then multiplied with the weight of the sample's class. If the model is far from correct on a minority sample, the loss thus increases more than if it was a majority one. The derivative of the loss function is changed similarly, and therefore the size of the step taken in the gradient descent. The second case usually implies oversampling (duplicating) the small classes or undersampling (removing some samples of) the larger classes. The two methods approximately do the same thing, as training on two equal samples in a row is very similar (only slightly more granular) to training on one but updating the weights twice as much.

2.3.3 Batch Normalization

During training of an ANN, all weights are typically updated simultaneously, after each batch. This means that the input to any non-input layer can have wildly different distributions after each weight update (if the distribution of the input of the new batch is not such that it counters this, but let us assume that all batches have the same input distribution). This slows the training process, especially for deep networks. The phenomena was referred to as "internal covariate shift" by Ioffe et. al. in their 2015 paper offering a solution to the problem [29]. They solve it by normalizing each batch to zero-mean and unit variance, then scaling and translating by trainable parameters, as following:

$$\begin{cases} \hat{x}^{(k)} = \frac{x^{(k)} - \mu_B^{(k)}}{\sigma_B^{(k)}} & (Normalization) \\ y^{(k)} = \gamma^{(k)} \hat{x}^{(k)} + \beta^{(k)} & (Scaling\, and\, translating), \end{cases} \tag{3}$$

where k can be any of the dimensions of the input, $x^{(k)}$ are the inputs of the (so called, colloquially) BatchNorm layer, $\mu_B^{(k)}$ the batch mean, $\sigma_B^{(k)}$ the batch standard deviation, $y^{(k)}$ the output, and $\gamma^{(k)}$ and $\beta^{(k)}$ the trainable scaling and translation parameters, respectively. During inference, instead of using the batch mean and standard deviation, the mean and standard deviation of the entire training set (roughly, the interested reader is referred to TensorFlow's web page on the topic [30]) is used.

Some [31] argue that the reason that batch normalization works is not because it reduces internal covariate shift, but rather because it "makes the optimization landscape significantly smoother", thereby inducing "a more predictive and stable behavior of the gradients, allowing for faster training". Why batch normalization works is neither within the scope of this thesis, nor relevant to its findings, so this debate will remain uncommented on. What needs to be understood is *that* it works.

3 Method

This part of the thesis describes how the theories introduced in the last chapter are implemented to solve the problems stated in the introduction.

3.1 Data preparation

3.1.1 Cleaning

The first task in many machine learning projects is to clean the data to suit the problem one wishes to solve. So too, begun this project.

The goal here was to transform data described in Section 1.3 to data sets where each keystroke correspond to a 6-dimensional (3 dimensions each for both the accelerometer and the gyroscope) time series (input) and a label representing the key pressed (output). The exact details of this process are not of interest, but due to reasons of reproducibility and transparency, the data removed for practical reasons will be presented here.

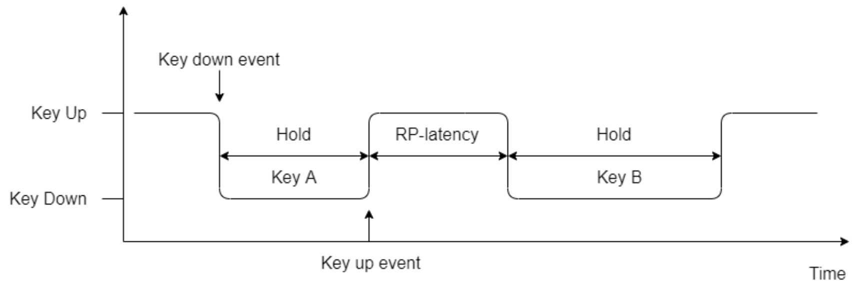


Figure 6: Two different keys being depressed at different times, with relevant keystroke dynamics nomenclature.

When typing quickly, the key-down event of a keystroke can sometimes occur before the key-up event of the previous keystroke, that is, one starts pressing one button before releasing the previous one (in Figure 6 seen as a negative RP-latency, and the keystrokes intersecting). In such a scenario, the keystrokes may interfere with each other, potentially tainting them both. To not make the problem unnecessarily difficult, keystrokes of both types were removed.

Another problem that arises in the BB-MAS data set is when the hold time for a keystroke is too long, the key-down event is registered multiple times. For the same reason as above, keystrokes with this behaviour were simply removed.

Non-standard characters were also removed, the characters kept for the **backspace problem** were all lower-case letters, all numbers, period ('.'), space (' '), new line/enter, and backspace. For the **general problem**, keys considered too rare (with less than 500 instances in the the remaining data set) were also removed due to fears of the class imbalance becoming too extreme. The remaining characters are period ('.'), space (' '), new line/enter, backspace and all lower-case letters except 'z'.

The last class of keystrokes removed were those with hold times larger than 170 ms, to deal with the problem with samples of varying length, described in Section 2.3.1.

To deal with the irregular sampling discussed in Section 1.3, accelerometer and gyroscope measurements of the same timestamps were averaged to get the final value, and to get a time series with equally distanced data points, they were linearly interpolated to get one data point for each millisecond.

Next, for each keystroke, the accelerometer and gyroscope measurements from 15 ms before the key-down event to 15 ms the after key-up event was chosen as the feature for that keystroke. Thus, as the hold times of all samples were smaller than 170 ms, each keystroke corresponds to a time-series of 200 time steps or less.

The label of each sample was encoded to 1 (Backspace) or 0 (Not backspace) in the backspace problem. In the general problem, keys were encoded to any non-negative number up to 28, then One-Hot-Encoded [32], yielding a 28-dimensional vector with 1, in one spot and 0 in all the rest for all samples. A sketch of the input and output of a model can be seen below in Figure 7.

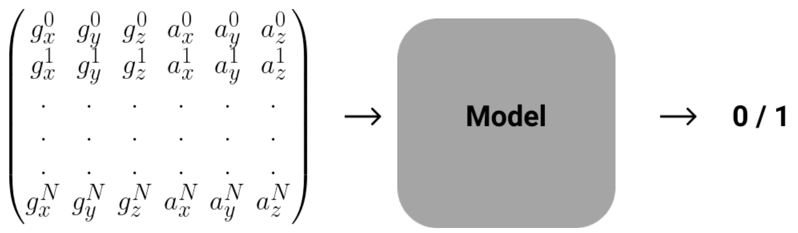

g_x^0 g_y^0 g_z^0 a_x^0 a_y^0 a_z^0
g_x^1 g_y^1 g_z^1 a_x^1 a_y^1 a_z^1
g_x^N g_y^N g_z^N a_x^N a_y^N a_z^N
Model
0 / 1

Figure 7: Sketch of the output/ input of the models created in this thesis. The input is a matrix of $6 \cdot N$ elements, where N is the length of the input and $N = T_{hold} + 2 \cdot 15$. g refers to gyroscope, a to accelerometer data. a_x^0 refers to the accelerometer measurement in the x-axis at 15 ms before the key-down event, a_x^N refers to the same 15 ms after the key-up event. The output is either 0 or 1 (as in the figure) in the backspace problem or any non-negative integer up to 28 in the general problem.

3.1.2 Data split and general process

The two problems, backspace and general, were solved in the same three cases, introduced in Section 1.2. Before going into the details of exactly how these were applied in practice, they are presented summarized below:

- **General case**: Divide the users into training and testing users. Divide the training users' data into a training and a test set. Train a model on the training set. Test the model on both the training users' test set and on the test users.
- **Brand-specific case**: Divide the users into the HTC and Samsung sets, depending on the phone model they used. For each brand, do as in the general case.
- **User-specific case**: Divide the users into training and testing users, then divide both training and testing users' data into a training and a test set. Use most of the training users' training set to build a "base" model. For each user, both training and testing, fine tune this base model using (in the case of the training users, the rest of) that user's training set, using

transfer learning. This yields one model per user. Test the models on each user's test set.

Now, for the more detailed description of and motivation for the cases.

In the User-specific case, we wish to build one model for each of the 116 users of the training set. For reasons described in Section 2.2, instead of simply training these models from the ground up, a base model was first trained on data coming from most users, and then fine tuned individually for each user, by means of transfer learning. This results in one model for each user, with a "base" trained on most users, and fine tuned using only that user's data. To accommodate this, and make sure not to reuse data or mix training and test data, a data split scheme was devised, seen in Figure 8.

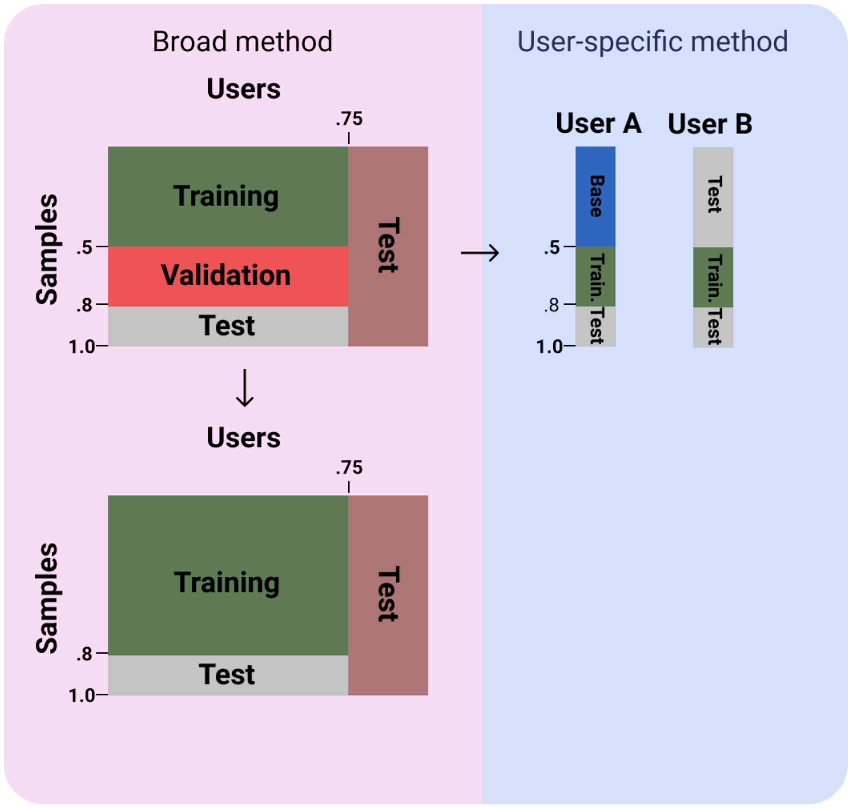


Figure 8: The data split scheme used in this thesis. The numbers represent fractions used in the different sets. User A is an example of a training user, User B of a test user.

The first thing to note is that the users are divided in training and testing users, with 75 % of users going into the training set and the remaining to testing. The training user set was then divided further into training (50 %), validation (30 %) and testing (20 %). Using the validation set to optimize, a base model to be used in the transfer learning was trained on the training set. The validation set was later used as the training set for each training user (in Figure 8 exemplified by User A) in the user-specific case. An equal fraction, 30 %, was used as the training set for the test users (exemplified by User B). After the final training is done, this leaves the training users with 20 % left to test on (since we do not want to test on data used for training, even of the base model), the testing users, for whom no data has been used to train the base model, 70 %. This creates two distinct groups of users, for which one would expect different results. Both are fine tuned in the same way, but only one group is represented in the base model. More on how the transfer learning was performed will be presented later on, in Section 3.2.2

In total, the users in the training set had 80 % of the their data used for training, while the test users only had 30 %. This is motivated by the wish for the ratio of data used in the transfer learning to be equal for both groups, but there is also an argument to be made for both groups having equal ratios of data used for training in total. Therefore, another experiment was set up where the test users used 80 % of their data for the transfer learning.

Going back to the general case, after determining the best base model, the validation set was merged into the training set, then the model was retrained on this larger training set. This maximizes the amount of data used for training. Here too two distinct groups of users can be found, one represented in the model, one not. Note though that none of the *data* is used for both training and testing, but some *users* are. All users were tested separately, so the results can easily be split up into users that the model has seen, and those that the model has not seen.

In the brand-specific case, the users were first divided into HTC and Samsung sets, then each of those into training and testing users. The optimal model structure found in the general case was chosen as the structure in this case as well. As the only use of the validation set was to infer optimal model structure, there was no need for it. Instead, the validation set was directly merged into the training set. Thus, for each brand the data was divided as what can be seen in the lower part of Figure 8.

For each of the problems, general and backspace, we now have four different cases, each yielding a result. These are the **general** and the **brand-specific** cases, and the two **user-specific** cases, one having the testing users train on 30 % and one on 80 % of their available data. For each of the cases, the results can be further divided into those coming from users the model has trained on (seen users), and those it has not (unseen users).

3.1.3 Further preparations

In agreement with the discussion in Section 2.3, time series shorter than 200 steps were zero-padded to 200 steps to let the models train efficiently.

To deal with the issues of imbalance discussed in the same section, different measures was taken in the two different problems. In the backspace problem, the positive samples, which were much fewer than negative, were upsampled (duplicated) until the training set contained half negative and half positive samples. Note that no upsampling was done on the test sets.

In the general problem, the remaining classes after removing the rarest ones were still vastly different in size. Upsampling all the small classes would thus artificially increase the size of the training set too much, thereby increasing the training time beyond what can be considered acceptable. Therefore, class weights were used instead.

3.2 Backspace problem

The backspace problem, where gyroscope and accelerometer streams are to be classified as either not backspace (0) or backspace (1), is in its nature a binary classification problem. The different models built to solve the problem in the three different cases are presented below.

3.2.1 General and Brand-specific cases

After experimenting with many different structures, details can be found in Appendix B, the best performing model had four bidirectional LSTM layers, followed by an output node, with a batch normalization layer between each layer. The LSTM layers had $25 \cdot 2$ (due to bidirectionalality) units each, with activation functions as implied by Figure 4. The output activation function was the sigmoidal function. The loss function used to train the model was binary cross-entropy. A small set of the training data was used to validate the model during training, after each epoch. From this validation result the optimal number of epochs could be inferred.

Note that a masking layer, or anything replicating its behaviour, is not included in the model. Including such a layer was tested, but performed worse (and caused the model to take substantially more time to train). Why it performed worse is unknown.

In the brand-specific case, the same model structure was used, the difference being the number of epochs the model trained for, inferred like in the general case.

3.2.2 User-specific case

To do the transfer learning, all layers down to the last LSTM layer were frozen, leaving the dense layers (and their accompanying BatchNorm layers) to be retrained. These layers were randomly initialized, rather than reusing the weights from the base model as the starting point. Note that the structure of the models were remained equal to that of the base-model. The "optimal" number of epochs was inferred independently for each user, in the same manner as in the broad case.

3.3 General problem

For the general problem, the multiclass classification problem where the samples are to be classified as (almost) any standard character on the keyboard, the final model was similar to that of the backspace problem. It had four BLSTM layers, followed by two dense layers, with a batch normalization layer between each layer. The BLSTM layers had $25 \cdot 2$ units each, and the two dense layers had 100 and 29 (the number of possible keys) nodes. The LSTM layers had activation functions as implied by Figure 4, the first dense layer ReLU, and the output had softmax as activation. The loss function used to train the model was categorical cross-entropy.

Class weights are usually calculated as [33]

$$w_i = \frac{N}{N_c \cdot n_i}, \tag{4}$$

where w_i is the class weight of class i, N the total number of samples, N_c the total number of classes, and n_i the number of samples of class i. Training with this definition of class weights yielded worse results on all aggregate metrics (but better on metrics pertaining to some rare characters) when compared to training without class weights. Training without class weights made the models too biased though, so as a compromise the class weights were calculated as

$$\hat{w}_i = \sqrt{w_i} = \sqrt{\frac{N}{N_c \cdot n_i}}. \tag{5}$$

The resulting models are still biased towards common classes, but less so than those that train without class weights. To what degree this is a problem depends on the use of the model.

How the number of optimal epochs was inferred and how the brand- and user-specific cases were implemented, was the same as for the backspace problem, see Section 3.2.1.

4 Results

4.1 Backspace problem

The results of the backspace problem will be presented with tables of summary statistics, confusion matrices [34] and a score-histogram describing the distribution of the model output scores for both positive and negative samples. In general, only results relevant to the analysis will be presented in the main part of the thesis, full results can be found in Appendix A.

4.1.1 Metrics

In Tables 4 and 5, the abbreviated metrics are, with links to further reading:

- **Acc**: Accuracy, ratio of correctly classified samples. When classifying between imbalanced classes, accuracy may be misleading, as the majority class is in total given undue weight when each sample is given equal weight. The accuracy should thus not be the only metric on which to judge the models. More discussion on the problem with accuracy can be found in Section 2.3.2.
- **Prec**: Precision [35], ratio between true positives and all *predicted* positives. Using Figure 9, $Prec = \frac{TP}{TP+FP}$. A perfect score near 1 can often be achieved by only classifying the most likely sample as positive, why precision should not be used alone.
- **Rec**: Recall [35], ratio between true positives and all *actual* positives. Using Figure 9, $Rec = \frac{TP}{TP+FN}$. A perfect score of 1 can be achieved by classifying all samples as true, recall should thus also not be used alone. There is often a trade-off between precision and recall, regulated by for example the classification threshold. Suppose for simplicity that the true classes are balanced. Decreasing the threshold would then increase both TP and FP, with FP growing faster relative to its size if the model has any classifying power. This leads to the recall increasing and the precision decreasing. Increasing the threshold would similarly decrease the recall and increase the precision. Precision and recall used together can thus be much more useful than either of them on their own.
- **F1**: F1-score [36], the harmonic mean of the precision and the recall. Summarizes precision and recall in one metric, where a value of 1 means that both precision and recall are 1, and 0 that either (or both) of them are 0.

- **EER**: Equal Error Rate, the rate at which the False Positive Rate ($FPR = \frac{FP}{FP+TN}$) and False Negative Rate ($FNR = \frac{FN}{FN+TP}$) are equal. Increasing the classification threshold increases the FNR, as more samples are considered negative, and decreases the FPR. Imagine sliding the threshold until $FPR = FNR$, then $EER = FPR = FNR$. The EER is unbiased regarding class imbalance, and gives the negative and positive class equal importance. It obviously does not take the model's classification threshold into account, but rather describes the rest of the model.

- **AUC**: Area Under the ROC Curve [37]. The ROC curve plots the true positive rate against the false positive rate for different thresholds. A value of 1 is perfect, 0.5 indicative that the model is just guessing at random, and 0 that it classifies wrong in every case, regardless of threshold. The AUC is equal to the probability that the model scores a randomly chosen positive sample higher than a randomly chosen negative one, and is thus not overly biased towards either positive or negative samples. The AUC, too, does not take the classification threshold into account.

Note that all metrics can take values between 0 and 1, and for all except the EER, a high value is good. Depending on the purpose of the model, different metrics should take center stage. The F1-score and EER are good metrics for when the positive and negative classes are of equal importance. Precision is a good metric if the cost of incorrectly classifying negative samples is much higher than the cost of classifying positive ones, recall if the opposite is true. In general, the F1-score and EER is probably best indicators of a good model in this case.

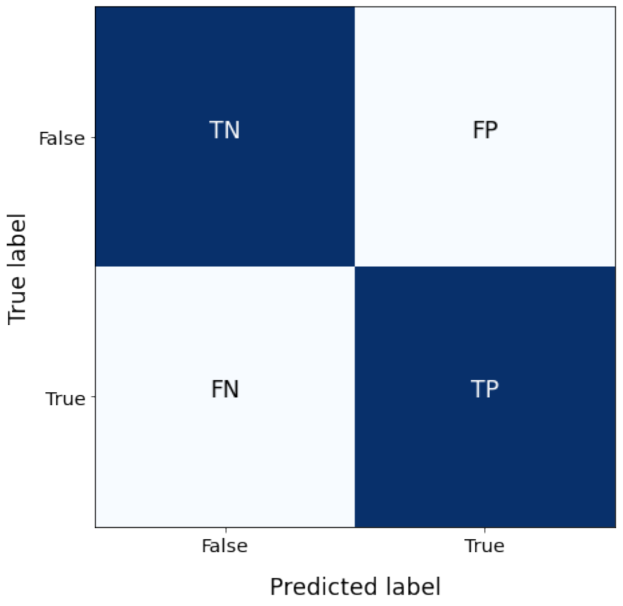


Figure 9: Anatomy of a confusion matrix, not normalized. TN: True Negatives, FP: False Positives, FN: False Negatives, TP: True Positives.

4.1.2 Results

The difference in results between the users whose data the models had partly seen and those the models had not seen was qualitatively similar in all cases, with the models always performing slightly worse on unseen users for all metrics. Below, in Table 4, the results for the general case, where one model trained on many users, are presented for both types of users. For the results from the other cases, see Appendix A.

The results may be difficult to interpret on their own, with nothing to compare them to. One can note though that they are better than randomly guessing, which means that the models have *some* classifying power. Also of interest is the difference between the different types of users (and cases, see Table 5), which certainly can be seen in the tables. More discussion on the results can be found in Section 5.

Table 4: Model results for users whose data the model has both trained on (**Seen**) and not (**Unseen**) in the backspace problem, in the general case. Other methods had qualitatively similar results.

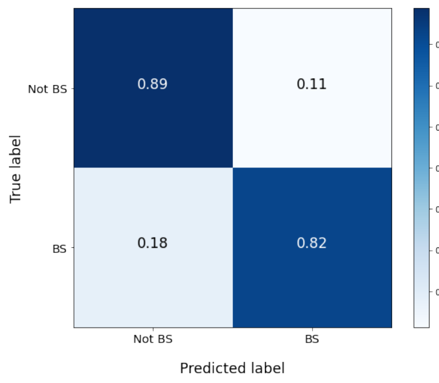
	Seen	**Unseen**
Acc	0.89	0.87
F1	0.64	0.56
EER	0.10	0.16
AUC	0.96	0.91
Prec	0.49	0.44
Rec	0.91	0.77

The table below presents the total results, on both seen and unseen users, in all the different cases. In the user-specific case, where one model was built for each user, unseen only refers to the *base*-model not having trained on that user's data, transfer learning was performed on all users. The results of the different brands in the brand-specific case, where one model was built for each of the two brands, were similar (details in Appendix A), and are merged for all results below.

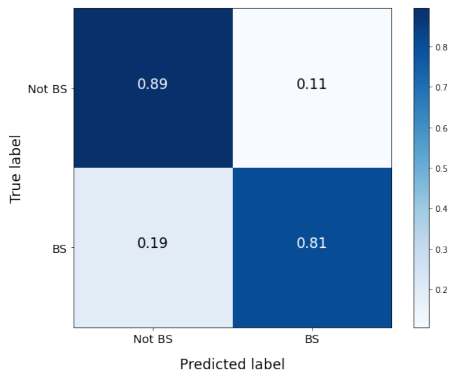
Table 5: Results of the backspace problem, in each of the three cases, for seen and unseen users combined. The best result of each metric is in **bold**. As there were two different ways to divide the test user data in the user-specific case, the results is divided as well. **User 30** refers to the results where the test (unseen) users trained on 30 % of their data, in the **User 80** column the test users had trained on 80 %. Note that the only difference between the two columns are due to the unseen users, the seen users are unaffected by this.

	General	**Brand**	**User 30**	**User 80**
Acc	0.88	**0.88**	0.87	0.88
F1	0.59	0.60	0.59	**0.61**
EER	0.14	0.14	0.12	**0.11**
AUC	0.93	0.93	0.94	**0.95**
Prec	0.46	**0.48**	0.44	0.46
Rec	0.82	0.81	0.89	**0.90**

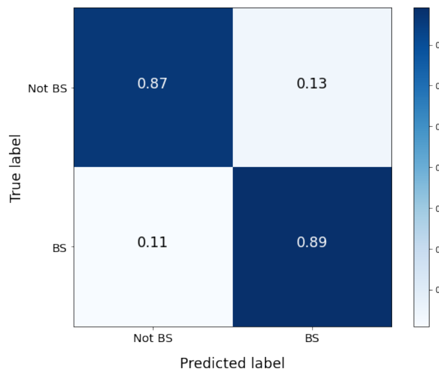
The figure below show confusion matrices in all different case. Note that the cells are normalized as ratios of the size of the actual (true) classes.



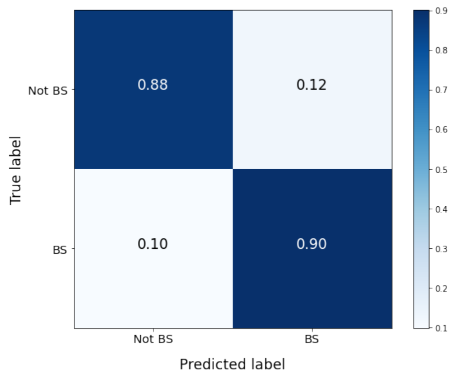
(a) General case



(b) Brand-specific case



(c) User-specific case, with the test users' models training on 30 % of their data.



(d) User-specific case, with the test users' models training on 80 % of their data.

Figure 10: Confusion matrices for the different backspace problem cases, with ratios normalized within each true label (each row sums up to one).

The backspace problem models all output a number between 0 and 1, where if the model was perfect all non-backspace samples would receive a score close to 0 and all backspace samples a score close to 1. Below are the output score distributions for both positive and negative samples from the general case for the backspace problem. The other models had qualitatively similar distributions.

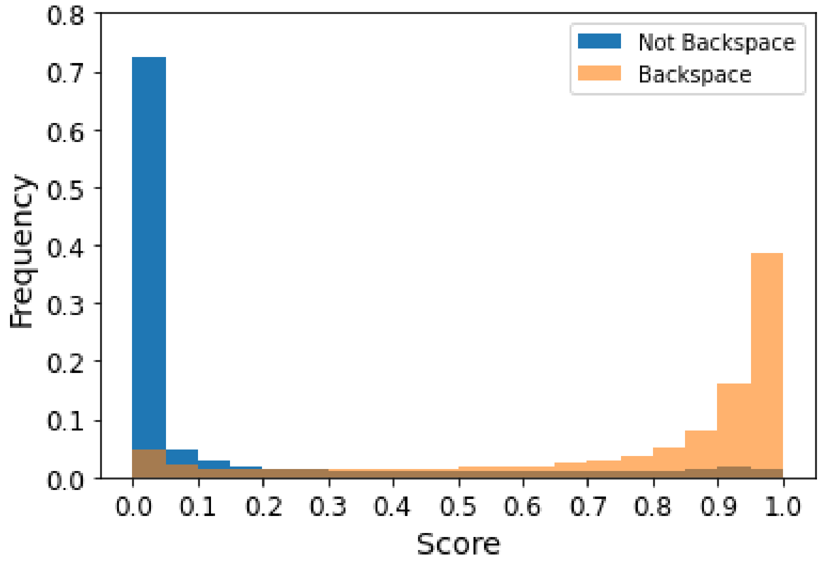


Figure 11: Distribution of output scores of the backspace problem general model. If the model were perfect, the figure would just show a blue bar to the furthest left and an orange bar to the furthest right. Note that the distributions are normalized independently (for example, roughly 40 % *of the positive samples* receive a score between 0.95 and 1.0).

4.2 General problem

In contrast to the backspace problem, the general problem is a multiclass classification problem. This calls for different metrics when evaluating model results. The result will be presented using tables of metrics (the interpretation of which will be explained in the following section) and a heatmap describing the multivariate confusion matrix. Here too, only results relevant to the analysis will be presented, for more results, see Appendix A.

4.2.1 Metrics

The metrics used in Tables 6 and 7 are:

- **Acc**: Accuracy, ratio of correctly classified samples over all classes. Note that the problem of imbalance encountered in the backspace problem is only exacerbated by the number larger number of classes and disparity between sizes of the classes.
- **F1**: Macro averaged F1-score [36] (in [38] called Averaged F1), calculated by taking the arithmetic average of the F1-scores of all classes. The F1-

score of a class is calculated by considering that class as positive, and all other classes as negative, thereby "collapsing" the multiclass confusion matrix to the binary one found in Figure 9, then calculating F1 as one would in the binary case, see Section 4.1.1. The macro averaged F1-score is not biased towards larger classes since it is the *unweighted* average of class F1 scores, it thus considers small classes to be of equal importance as large ones.

- **MR**: Mean Rank of all test samples. For each test sample, sort the outputs from most likely to least likely (according to the model), then find what "rank" the true class placed as. For example, if the model considers the true class of the sample to be second most likely, the rank of this sample is 2 (the rank is thus not the *index* of the true class in the sorted output (1), but rather what is usually considered the rank) The mean rank is the arithmetic mean of all test samples' rank. It ranges from the best possible outcome, 1, where all samples are correctly classified, to 29, where the true label is considered least likely for all samples (incidentally, such a model would be equally useful as the best possible one, as one could simply revert its result, yielding perfection). The mean rank is biased towards models biased towards larger classes, as biasedly classifying as a large class can artificially drive the mean rank down. Imagine trying to infer a single key using the method presented in this thesis. The mean rank is roughly how many times one would have to try to get the correct character (given that the characters in the password are distributed similarly to the characters in the test set).

- **Acc3**: Accuracy if the true class being in the "top three" is considered to be correct, using the same ranked list of outputs as in **MR**. Imagine a password of length N. Trying three times per character, the model correctly guesses this password in 3^N guesses with a probability of $p = (Acc3)^N$ (once again, given that the characters in the password are distributed similarly to the characters in the test set).

- κ: Cohen's Kappa [39], a metric comparing the agreement between predictions and actual classes and the agreement that would arise by pure chance. It is calculated as

$$\kappa = \frac{p_o - p_e}{1 - p_e},$$

where p_o is the observed agreement (the accuracy of the model), and p_e is the agreement that would occur "by chance". p_e is calculated as

$$p_e = \sum_k p_{\widehat{k,pred}} \cdot p_{\widehat{k,true}} = \frac{1}{N^2} \sum_k n_{k,pred} \cdot n_{k,true},$$

where $p_{\widehat{k,pred}}$ is the probability that a given *predicted* label is k, $p_{\widehat{k,true}}$ is the probability that a given *actual* label is k, with the probabilities coming from the distributions of the actual and predicted labels, N is the total number of observations, $n_{k,pred}$ the number of predictions of class k, $n_{k,true}$ the number of samples that are actually of class k. Imagine, already given the distribution of the predicted and true labels, assigning a true and predicted label independently randomly, with probability distributions from the given distributions. The expected agreement between the true and predicted labels (accuracy of this "naive model") is then p_e. The fraction defining κ is the ratio between the difference between the accuracy and the accuracy "expected by chance" and the difference between the best possible accuracy (1) and the same. The best value arises when the accuracy is 1, $\kappa_{max} = 1$, if the predictions is equal to what would occur by "chance" (given the prediction distribution) $\kappa = 0$. If the model performs worse than chance, the value may be negative, with the worst case scenario being $\kappa_{min} = -1$. Since κ takes the expected chance agreement into account it is robust to class imbalance, and thus more useful for our case than for example accuracy.

- **AUC**: Macro average of the Area Under the ROC curve of all pairwise combinations of classes. For each combination (where $\{a, b\} \neq \{b, a\}$), consider one of them as positive and the other as negative, calculate the AUC as in the binary case (see Section 4.1.1), then take the average for all pairs of classes. This way of calculating the AUC is "insensitive to class imbalance" [40], and therefore useful for our problem. More information can be found at [41] and [42].

Here too, different metrics are better for different purposes of the model. The F1-score and κ are best if all classes are of equal importance, accuracy and mean rank if the importance of a class is proportional to its frequency in the test data.

4.2.2 Results

Below are the results for the seen and unseen users in the general case of the general problem, using the metrics explained above. The other methods had qualitatively similar results. The results here are also difficult to interpret on their own, but they are certainly better than randomly guessing. More analysis of the results can be found in Section 5.

Table 6: Model results for users whose data the model has both trained on (**Seen**) and not (**Unseen**) in the general problem, in the general case. Other cases had qualitatively similar results.

	Seen	**Unseen**
Acc	0.49	0.39
F1	0.32	0.23
MR	2.73	3.46
Acc3	0.78	0.68
κ	0.45	0.34
AUC	0.92	0.88

Below are the total results for all different cases for the general problem.

Table 7: Results of the general problem, in each of the case, for seen and unseen users combined. The best result of each metric is in **bold**. As there were two different ways to divide the test user data in the user-specific case, the results is divided as well. **User 30** refers to the results where the test (unseen) users trained on 30 % of their data, in the **User 80** column the test users had trained on 80 %. Note that the only difference between the two columns are due to the unseen users, the seen users are unaffected by this.

	Broad	**Brand**	**User 30**	**User 80**
Acc	0.43	0.43	0.49	**0.51**
F1	0.26	0.27	0.30	**0.32**
MR	3.19	3.37	2.93	**2.80**
Acc3	0.72	0.71	0.78	**0.79**
κ	0.38	0.38	0.45	**0.47**
AUC	**0.90**	0.89	0.88	0.89

The heatmap below describes the confusion matrix for all labels examined in the general problem, solved in the general case. The colormap is truncated (some of the cells that seem to be of value 0.30 are rather larger than 0.30) to make the low values easier to distinguish from each other, the "true" heatmap with the full colormap range can be found in Appendix A. Also note that the values are normalized by true label, each cell $p_{k,j}$ describes the ratio $\frac{n_{k,j}}{n_j}$, where $n_{k,j}$ is the number of samples where the predicted label is k and the true label is j, and n_j is the number of samples where the true label is j. Each row then sums up to 1.

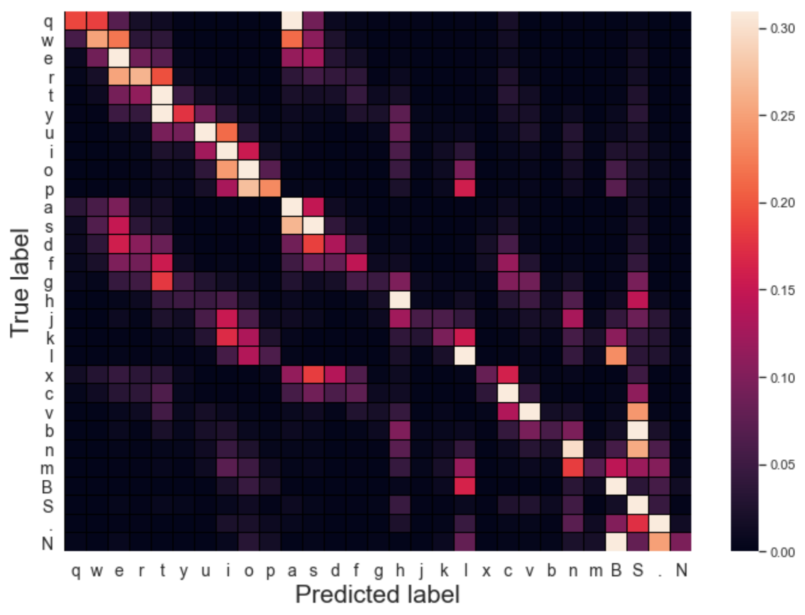


Figure 12: Heatmap describing the distribution of predicted labels for each true label in the general problem, in the general case. The cells are normalized by each true label, and the colormap is truncated (for a heatmap with an untruncated colormap, see Appendix A), some cells labeled as 0.30 are rather larger than this. The last characters in the list are 'B': Backspace, 'S': Space, '.': Period, and 'N': New line/Enter.

5 Analysis

5.1 Naïve model results

To interpret the model performance, it helps to compare it to some baseline performance. For this purpose, three naïve models have been created for each of the two problems. They behave as follows for both problems:

- Model 1: Always classify as the largest class.
- Model 2: Classify samples randomly, with probability distributions equal to that of the training data.
- Model 3: Classify samples randomly, with uniform distribution.

Obviously, these models should all have atrocious performance, but are useful to confirm that our models are at least better than the least laborious models possible. Note that metrics taking the output score of a model into account, rather than the output of the classification thresholding, will not make any sense for the naïve models. They will thus not be presented.

5.1.1 Backspace problem

Below are the results for the three naïve classifiers for the backspace problem.

Table 8: Naïve model results for the backspace problem. The numbers at the head of the table refer to the models described above. Precision for Model 1 is undefined, as there are no true or false positives.

	1	**2**	**3**
Acc	0.89	0.81	0.50
F1	0.00	0.11	0.17
Prec	-	0.11	0.10
Rec	0.00	0.11	0.50

Comparing to Table 5, we see that our models indeed perform better than all naïve models, with a notable exception in the accuracy metric. This, performing worse on the accuracy metric, may seem disappointing but does not necessarily mean that the models are worthless (though it may be considered a symbolic defeat). The question becomes, inspired by the discussion in Section 2.3.2, how many false positives are we willing to tolerate for each false negative? Depending on what the use of the models is, the answer will vary. The models may be changed by adjusting the classification threshold according to one's need (the best possible accuracy score attainable by this method is roughly 92 %, with a threshold of 0.92). One could also have adjusted the cardinality ratio of the training data, by up-/down-sampling, to suit the relevant problem. If the training was done on a training set with the "natural" ratio (roughly 11 % backspace) the accuracy would certainly improve, though to the detriment of the EER and recall.

In the end, a classification threshold of 0.5 was chosen since there are arguments both for decreasing and increasing it from this "neutral" state. Looking at Figure 11, we see that changing the threshold slightly (between 0.3 and 0.65) would not affect the model performance too much, as the distributions are separated enough. Changing the threshold dramatically (such as to 0.92 as in the example above) would not make much sense either, at least if *all* backspace samples together are given roughly as much importance as *all* non-backspace samples together. If such is the case, Figure 11 is reasonable to study, as opposed to a similar one where the distributions are not normalized independently.

5.1.2 General problem

Below are the results for the naïve models in the general problem.

Table 9: Results of naïve models in the general problem. The numbers at the head of the table refer to the models described above.

	1	**2**	**3**
Acc	0.16	0.07	0.03
F1	0.01	0.03	0.03
κ	0.00	-0.00	0.00

Comparing to Table 7, we see that the naïve models are vastly outperformed by our models.

5.2 Seen versus Unseen

Looking at both Tables 4 and 6 we see, unsurprisingly, that models perform better on seen users than unseen ones. This implies that there is some dissimilarities between users. Notably though, the difference in performance is somewhat small, indicating that the users are not *completely* dissimilar. One can conclude that it is possible to train a model on many users and use it for other users, but the results are slightly worse for the unseen users.

5.3 The different cases

Before analysing the results yielded in the different cases, let us quickly state how one would expect them to compare.

In the User- and Brand-specific cases, more focus is put on what is believed to be more relevant data, and irrelevant data is not allowed to dilute the data as much. On the other hand, the models do not get to train on as much data. This leads to a trade-off between how large and how specific the training set should be. The question becomes "how similar are the samples coming from different users/brands, especially compared to those from the same user/brand"? Enough that different users/brands can be used for the same model? Or dissimilar enough that they are best left divided? The results could go either way.

5.3.1 Backspace problem

Looking at Table 5 we see that the results do not differ much at all between the different cases. While the user-specific (80) models slightly outperform the rest on most metrics, the differences might be due to the stochastic nature of the training of ANNs. We see that between the user- and the brand-specific models, the user models seem to be more balanced, less prone to blindly favor one class over the other. The brand models favored the majority class (not backspace), why their accuracy score was better, at the cost of their recall score. This issue of balance is most easily seen in the confusion matrices 10b and 10d, where the absolute difference between the FPR and the FNR (on the anti-diagonal) is larger for the brand models than for the user ones. A completely balanced model has equal FPR and FNR. Note that this difference probably is due to random coincidence, rather than some inherent characteristic in the data or the way the models were set up. Since all models were trained on balanced data, one would expect the models to all be somewhat balanced, which is true enough.

5.3.2 General problem

In the general problem, the difference between the different cases is much larger. While the general and the brand-specific results are similar, the user-specific models are substantially better. It is difficult to ascertain *why* this is, one theory is that as the general problem is much more difficult, using specific data for training is necessary. A counter-argument could be made, though, that as it is much more difficult, using *more* data for training is necessary. Perhaps the method in the user case is "better" for both problems, but the models reach an impasse in the backspace problem, where further improvements are much harder to reach. The user models are better across the board, except notably for the AUC, where the general model slightly outperforms all other models. Why this is, is also a mystery.

This is not to say that the user-specific case *always* yields better results in the general problem. With less data per user it is certainly possible that the general model passes the user-specific ones in performance. Similarly, if more data was available per user/brand, one would expect the user/brand-specific models to outperform the general one even more. Even more so, in the case of the brand models, if the phone models are less similar (looking back at Table 1 we are reminded that the Samsung and the HTC phones used in this data-set are quite similar). This would be especially true if the phones were of different operating systems. When investigating swipes on the touch screen of Android and iOS cell phones, Cervin [43] found that they exhibit wildly different behaviours, and the same could certainly be true for keystrokes.

One could also have incorporated user and brand specificness by training a model for each category (user or brand) using data from all categories, but with higher sample weights for samples of the same category. This has scalability issues though, and would become unfeasible as the number of categories increase. One of the beauties of transfer learning, as done in the user-specific case, is that it vastly decreases the training time per category.

5.4 Heatmap

Figure 15, describing the predicted label distribution for each true label, shows that the model follows an intuitive behaviour. The most probable prediction for nearly all labels is the actual label (not for for example q though, this may be due to the model favoring large classes, like a, see next section), as shown by the lit up main diagonal. The cells on the *super-* and *subdiagonal* are also brighter than most, indicating that the model is prone to classify keystrokes as the key's horizontal neighbours. Note that this is not true for p and a, for example, who are only neighbours in the list of keys, but not actually on the keyboard. Similarly, looking some steps above/below the main diagonal we find that the model is also prone to classify keystrokes as their *vertical* neighbours, but since the keyboard is not completely rectangular these "diagonals" are somewhat wobbly. Note for example how the true label f is often classified as r, t and c.

5.5 General model performance on different characters

Tables 6 and 7 tell nothing about how models perform on different characters, only what the aggregate results are over all characters. Since the models used class weights calculated as in Equation (5), as the square root of the "conventional" class weights, one would expect the models to be biased towards the larger classes. Is this the case? Below, the worst, median, and best performing characters, F1-wise, are presented.

Table 10: General model performance on the characters' that got the best (**Space**), median (**r**) and worst (**x**) F1-results, in the general case.

	x	**r**	**Space**
F1	0.049	0.26	0.71
AUC	0.88	0.89	0.95
Prec	0.035	0.25	0.72
Rec	0.082	0.27	0.70

We see that they differ quite substantially on all metrics (save perhaps the AUC). To see whether this is due to x and r being less prevalent than *Space*, Figure 13, below, plots the F1 score of each character against the number of instances of that character in the training set for the broad model.

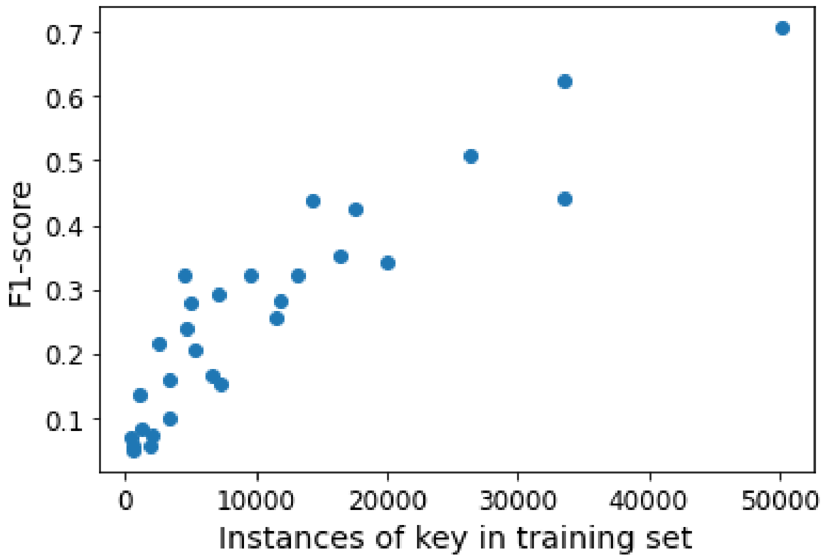


Figure 13: F1 score for each character plotted against the number of instances of said character in the training set for the general model, in the general problem.

We see that model performance on a character is indeed heavily correlated with its prevalence in the training set. The correlation coefficient is $\rho = 0.91$. The results would be more equally distributed for each letter if conventional class weights had been used, but as hinted in Section 3.3, this is not only due to the performance on rare characters increasing. The bulk of the decrease in performance variance between characters, when changing to conventional class weights, is explained by the performance on common characters deteriorating.

5.6 Future Work

While the results of this thesis are somewhat pleasing, this is certainly not the definitive paper on keystroke inference using accelerometer and gyroscope data. More work could be done, for example:

- The model hyperparameters used to build the models were chosen through somewhat haphazard experimenting. A more structured optimizing could have been done using for example Bayesian Optimization [44]. This would not necessarily yield the *optimal* hyperparameters (almost certainly not), but there is a large possibility that it would improve the models moderately.
- Most of the papers described in Section 1.4 use hand picked features from the motion sensors as input to their (comparatively smaller) models, instead of letting LSTM layers extract features directly from the motion

measurements. Perhaps the results could have been improved if the methods were combined, for example by letting the dense layers at the end of the models take both the output of the LSTM layers and the handpicked features used in the papers as input. This was done with the features hold time and RP-latency (both from the previous keystroke to the current one and from the current to the next, see Figure 6 and Appendix B), but this yielded worse results.

- A keylogger worth its name should certainly be able to infer all types of keys, but some keys were omitted from the general problem due to their infrequency in the data-set. If one were to find a larger and more diverse data-set, or handcraft one for this specific purpose, the general models could improve.
- The models in this thesis are not asked to detect when a keystroke occurs, only what key it is. In most real world settings (though not the one of interest to Callsign) one would also need to detect a keystroke occurring, *then* classify it.
- Instead of considering each keystroke in a vacuum, one could consider whole words or sentences, using words or legible sentences as a prior for the general model. For example, if model was highly certain that the last two keystrokes were `"c"` and `"a"`, the following keystroke is probably more likely to be `"r"` than `"Space"`, regardless of what the model in its current form says. This requires the corpus used for the prior distribution to be similar to that of the target purpose, so it needs to be chosen with care.
- It is difficult to compare the results of the general problem to that of other papers, as not only do they investigate other sets of characters (for example only numbers (PIN-codes) or the whole keyboard, without removing 'z' for instance), but also use other data-sets to train and test. To be able to compare the ability of LSTMs to classify keystrokes to other methods, one would need to both train and test on the same data-set.

5.7 Conclusions

The most important insight from this thesis is that it is undeniably possible for private information to leak through the motion sensors of an ordinary mobile phone. The quality of this information is not good enough for the LSTM models built in this thesis to perfectly infer text, but resoundingly better than guessing randomly. Furthermore, while knowing a specific person's typing pattern helps, it is possible to train a model on one population, then apply the model to another population, with only slightly worse results. This is obviously the most feasible way to build a model, as it may be difficult to get a potential user or target to agree to having their typing behaviour studied in detail.

In the backspace problem, where keystrokes are classified as either backspace or not backspace, the results are all very similar, regardless if the models are trained on many users and more than one phone model, on many users using the same phone model, or on a specific user. For the general problem though, where keystrokes are classified almost any general key on the keyboard, training one model specifically for each user is substantially better than the other methods. This can be done with transfer learning, which reduces the training time considerably. Note though that other methods can be better for other data-sets, depending on, among other things, the size of each user's data-set. The general models all suffer from being highly imbalanced, a consequence of the pronounced imbalance in the data-set used for training.

References

[1] A. K. Belman et al. *SU-AIS BB-MAS (Syracuse University and Assured Information Security - Behavioral Biometrics multi-device and Multi-Activity data from Same users) dataset.* URL: `https://ieee-dataport.org/open-access/su-ais-bb-mas-syracuse-university-and-assured-information-security-behavioral-biometrics`.

[2] A. K. Belman et al. *Insights from BB-MAS - A Large Dataset for Typing, Gait and Swipes of the Same Person on Desktop, Tablet and Phone.* URL: `https://arxiv.org/abs/1912.02736`.

[3] MathWorks. *Accelerometer.* URL: `https://www.mathworks.com/help/supportpkg/android/ref/accelerometer.html` (visited on 07/02/2021).

[4] A. Nahapetian. "Side-channel attacks on mobile and wearable systems". In: *2016 13th IEEE Annual Consumer Communications Networking Conference (CCNC).* 2016, pp. 243–247. DOI: `10.1109/CCNC.2016.7444763`.

[5] L. Cai and H. Chen. "TouchLogger: Inferring Keystrokes on Touch Screen from Smartphone Motion". In: *HotSec '11.* 2011. URL: `https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.359.9767&rep=rep1&type=pdf`.

[6] Z. Xu, K. Bai, and S. Zhu. "TapLogger: Inferring User Inputs on Smartphone Touchscreens Using on-Board Motion Sensors". In: *Proceedings of the Fifth ACM Conference on Security and Privacy in Wireless and Mobile Networks.* WISEC '12. Tucson, Arizona, USA: Association for Computing Machinery, 2012, pp. 113–124. DOI: `10.1145/2185448.2185465`.

[7] E. Owusu et al. "ACCessory: Password Inference Using Accelerometers on Smartphones". In: *Proceedings of the Twelfth Workshop on Mobile Computing Systems & Applications.* HotMobile '12. New York, NY, USA: Association for Computing Machinery, 2012. DOI: `10.1145/2162081.2162095`.

[8] L. Cai and H. Chen. "On the Practicality of Motion Based Keystroke Inference Attack". In: *Trust and Trustworthy Computing.* Ed. by Stefan Katzenbeisser et al. Springer Berlin Heidelberg, 2012, pp. 273–290. URL: `https://link.springer.com/chapter/10.1007/978-3-642-30921-2_16`.

[9] L. Wang. "I Know What You Type on Your Phone: Keystroke Inference on Android Device Using Deep Learning". MA thesis. University of Kansas, May 2019. URL: `https://kuscholarworks.ku.edu/handle/1808/29708`.

[10] S. Hochreiter. "Untersuchungen zu dynamischen neuronalen Netzen". MA thesis. Technische Universität München, Apr. 1991. URL: `https://www.researchgate.net/publication/243781690_Untersuchungen_zu_dynamischen_neuronalen_Netzen`.

[11] S. Hochreiter and J. Schmidhuber. "Long Short-term Memory". In: *Neural computation* 9 (Dec. 1997), pp. 1735–80. DOI: `10.1162/neco.1997.9.8.1735`.

[12] G. Chevalier. *LARNN: Linear Attention Recurrent Neural Network, CC BY-SA 4.0.* URL: `https://commons.wikimedia.org/w/index.php?curid=99599411` (visited on 08/26/2021).

[13] K. Greff et al. “LSTM: A Search Space Odyssey”. In: *IEEE Transactions on Neural Networks and Learning Systems* 28.10 (Oct. 2017), pp. 2222–2232. DOI: 10.1109/tnnls.2016.2582924.

[14] M. Schuster and K. K. Paliwal. “Bidirectional Recurrent Neural Networks”. In: *IEEE Transactions on Signal Processing* 45.11 (Nov. 1997), pp. 2673–2681. URL: https://maxwell.ict.griffith.edu.au/spl/publications/papers/ieeesp97_schuster.pdf.

[15] A. Graves and J. Schmidhuber. “Framewise phoneme classification with bidirectional LSTM and other neural network architectures”. In: *Neural Networks* 18 (5-6 2005), pp. 602–610. DOI: 10.1016/j.neunet.2005.06.042.

[16] J. P.C. Chiu and E. Nichols. “Named Entity Recognition with Bidirectional LSTM-CNNs”. In: *Transactions of the Association for Computational Linguistics* 4 (July 2016), pp. 357–370. DOI: 10.1162/tacl_a_00104.

[17] I. Habernal and I. Gurevych. “Which argument is more convincing? Analyzing and predicting convincingness of Web arguments using bidirectional LSTM”. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pp. 1589–1599. DOI: 10.18653/v1/P16-1150.

[18] H. Kang et al. “Time Series Prediction of Wastewater Flow Rate by Bidirectional LSTM Deep Learning.” In: *International Journal of Control, Automation and Systems* 18 (Dec. 2020), pp. 3023–3030. DOI: https://doi.org/10.1007/s12555-019-0984-6.

[19] N. Bhaskar, M. Suchetha, and N. Y. Philip. “Time Series Classification-Based Correlational Neural Network With Bidirectional LSTM for Automated Detection of Kidney Disease”. In: *IEEE Sensors Journal* 21.4 (2021), pp. 4811–4818. DOI: 10.1109/JSEN.2020.3028738.

[20] M. Khan et al. “Bidirectional LSTM-RNN-based hybrid deep learning frameworks for univariate time series classification”. In: *Journal of Supercomputing* 77 (Dec. 2021), pp. 7021–7045. DOI: https://doi.org/10.1007/s11227-020-03560-z.

[21] K. Simonyan and A. Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: *ICLR 2015*. 2015. URL: https://arxiv.org/abs/1409.1556.

[22] H. Jun et al. “Facial Expression Recognition Based on VGGNet Convolutional Neural Network”. In: *2018 Chinese Automation Congress (CAC)*. 2018, pp. 4146–4151. DOI: 10.1109/CAC.2018.8623238.

[23] P. Matlani and M. Shrivastava. “Hybrid Deep VGG-NET Convolutional Classifier for Video Smoke Detection”. In: *Computer Modeling in Engineering & Sciences* 119.3 (2019), pp. 427–458. DOI: 10.32604/cmes.2019.04985.

[24] M. Kim. “Contactless Palmprint Identification Using the Pretrained VGGNet Model”. In: *Journal of Korea Multimedia Society* 21 (12 2018),

pp. 1439–1447. DOI: `https://doi.org/10.9717/KMMS.2018.21.12.1439`.

[25] H. Ke et al. "Towards Brain Big Data Classification: Epileptic EEG Identification With a Lightweight VGGNet on Global MIC". In: *IEEE Access* 6 (2018), pp. 14722–14733. DOI: `10.1109/ACCESS.2018.2810882`.

[26] H. I. Fawaz et al. "Transfer learning for time series classification". In: *2018 IEEE International Conference on Big Data (Big Data)* (Dec. 2018). DOI: `10.1109/bigdata.2018.8621990`.

[27] TensorFlow. *Masking and padding with Keras*. URL: `https://www.tensorflow.org/guide/keras/masking_and_padding` (visited on 09/03/2021).

[28] Wikipedia. *Accuracy Paradox*. URL: `https://en.wikipedia.org/wiki/Accuracy_paradox` (visited on 09/09/2021).

[29] S. Ioffe and C. Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. URL: `https://arxiv.org/pdf/1502.03167.pdf`.

[30] TensorFlow. *Batch Normalization*. URL: `https://www.tensorflow.org/api_docs/python/tf/keras/layers/BatchNormalization` (visited on 09/02/2021).

[31] S. Santurkar et al. "How Does Batch Normalization Help Optimization?" In: *NeurIPS 2018*. URL: `https://arxiv.org/abs/1805.11604`.

[32] Wikipedia. *One-hot*. URL: `https://en.wikipedia.org/wiki/One-hot` (visited on 10/05/2021).

[33] scikit-learn. *sklearn.utils.class_ weight.compute_ class_ weight*. URL: `https://scikit-learn.org/stable/modules/generated/sklearn.utils.class_weight.compute_class_weight.html` (visited on 09/09/2021).

[34] Wikipedia. *Confusion Matrix*. URL: `https://en.wikipedia.org/wiki/Confusion_matrix` (visited on 09/07/2021).

[35] Wikipedia. *Precision and Recall*. URL: `https://en.wikipedia.org/wiki/Precision_and_recall` (visited on 09/07/2021).

[36] Wikipedia. *F-score*. URL: `https://en.wikipedia.org/wiki/F-score` (visited on 09/07/2021).

[37] Wikipedia. *Reciever Operating Characteristic*. URL: `https://en.wikipedia.org/wiki/Receiver_operating_characteristic` (visited on 09/07/2021).

[38] J. Opitz and S. Burst. *Macro F1 and Macro F1*. 2021. URL: `https://arxiv.org/abs/1911.03347`.

[39] Wikipedia. *Cohen's kappa*. URL: `https://en.wikipedia.org/wiki/Cohen%5C%27s_kappa` (visited on 09/07/2021).

[40] scikit-learn. *sklearn.metrics.roc_ auc_ score*. URL: `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score.html` (visited on 09/09/2021).

[41] scikit-learn. *Metrics and scoring: quantifying the quality of predictions*. URL: `https://scikit-learn.org/stable/modules/model_evaluation.html#roc-metrics` (visited on 09/09/2021).

[42] D. Hand and R. Till. "A Simple Generalisation of the Area Under the ROC Curve for Multiple Class Classification Problems." In: *Machine Learning*

45 (2001), pp. 171–186. URL: https://link.springer.com/article/10.1023/A:1010920819831.

[43] O. Cervin. “Authentication of Swipe Gestures in Smartphones using Sensor Data”. MA thesis. Lund University, 2019.

[44] J. Snoek, H. Larochelle, and R. P. Adams. *Practical Bayesian Optimization of Machine Learning Algorithms*. 2012. URL: https://arxiv.org/abs/1206.2944.

Appendices

A Full results

Only results relevant to the analysis was included in Section 4. This appendix includes more detailed results for the interested reader.

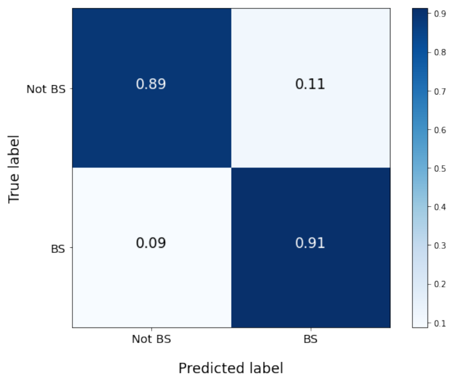
A.1 Backspace Problem

Table 11: Full results of the backspace brand-specific case, for both phone models and both seen and unseen users. **HTC** refers to the HTC-One, **Samsung** the Samsung Galaxy S6. The last column describes the results of all samples, both Samsung and HTC.

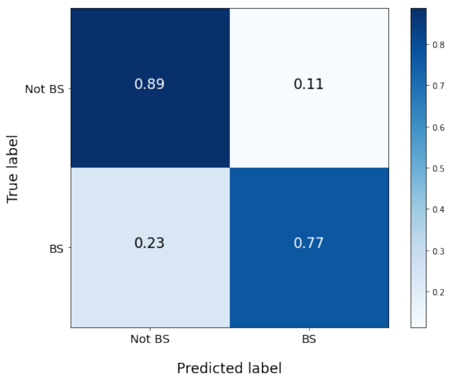
	HTC			**Samsung**			**Both**
	Seen	**Unseen**	**Total**	**Seen**	**Unseen**	**Total**	**Total**
Acc	0.90	0.86	0.88	0.92	0.87	0.89	0.88
F1	0.61	0.57	0.59	0.71	0.56	0.62	0.60
EER	0.11	0.17	0.15	0.09	0.17	0.14	0.14
AUC	0.95	0.91	0.92	0.97	0.91	0.94	0.93
Prec	0.47	0.46	0.46	0.59	0.44	0.50	0.48
Rec	0.86	0.76	0.80	0.89	0.76	0.81	0.81

Table 12: Results of the backspace user-specific case. As there were two different ways to divide the test user data, the unseen user's results is divided as well. **Unseen (30)** refers to the results where the test users trained on 30 % of their data, **Unseen (80)** to 80 %. Under the **Total (30)** column we find the results when **Seen** and **Unseen (30)** is combined, and similarly for **Total (80)**. Note that the **Seen** column is unaffected by this division.

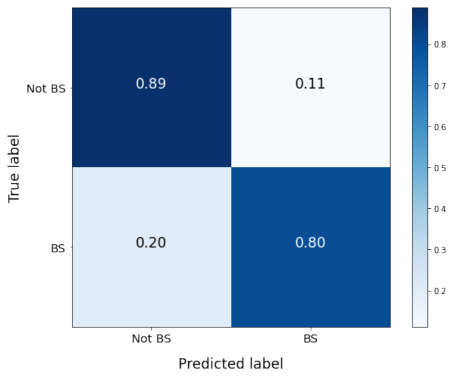
	Seen	**Unseen (30)**	**Total (30)**	**Unseen (80)**	**Total (80)**
Acc	0.88	0.86	0.87	0.86	0.88
F1	0.62	0.56	0.59	0.58	0.61
EER	0.11	0.13	0.12	0.13	0.11
AUC	0.95	0.93	0.94	0.94	0.95
Prec	0.47	0.41	0.44	0.43	0.46
Rec	0.90	0.88	0.89	0.89	0.90



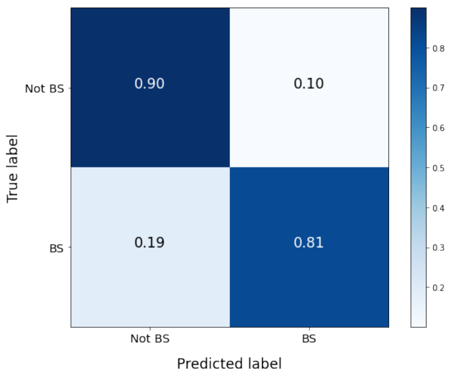
(a) Broad case, seen users.



(b) Broad case, unseen users.



(c) Brand-specific case, HTC users.



(d) Brand-specific case, Samsung users.

Figure 14: Confusion matrices for the backspace problem, for different cases and types of users, with ratios normalized within each true label (each row sums up to one).

A.2 General problem

Table 13: Full results of the general brand-specific case, for both phone models and both seen and unseen users. **HTC** refers to the HTC-One, **Samsung** the Samsung Galaxy S6. The last column describes the results of all samples, both Samsung and HTC.

	HTC			**Samsung**			**Both**
	Seen	**Unseen**	**Total**	**Seen**	**Unseen**	**Total**	**Total**
Acc	0.44	0.35	0.39	0.53	0.41	0.46	0.43
F1	0.27	0.19	0.22	0.37	0.27	0.30	0.27
MR	3.05	3.93	3.58	2.39	3.74	3.21	3.37
Acc3	0.73	0.63	0.67	0.82	0.70	0.75	0.71
κ	0.40	0.30	0.34	0.50	0.37	0.42	0.38
AUC	0.90	0.85	0.87	0.94	0.88	0.91	0.90

Table 14: Results of the general user-specific case. As there were two different ways to divide the test user data, the unseen user's results is divided as well. **Unseen (30)** refers to the results where the test users trained on 30 % of their data, **Unseen (80)** to 80 %. Under the **Total (30)** column we find the results when **Seen** and **Unseen (30)** is combined, and similarily for **Total (80)**. Note that the **Seen** column is unaffected by this division.

	Seen	**Unseen 30**	**Total 30**	**Unseen 80**	**Total 80**
Acc	0.51	0.48	0.49	0.50	0.51
F1	0.31	0.29	0.30	0.32	0.32
MR	2.82	3.02	2.93	2.72	2.80
Acc3	0.79	0.76	0.78	0.79	0.79
κ	0.47	0.44	0.45	0.46	0.47
AUC	0.89	0.87	0.88	0.91	0.89

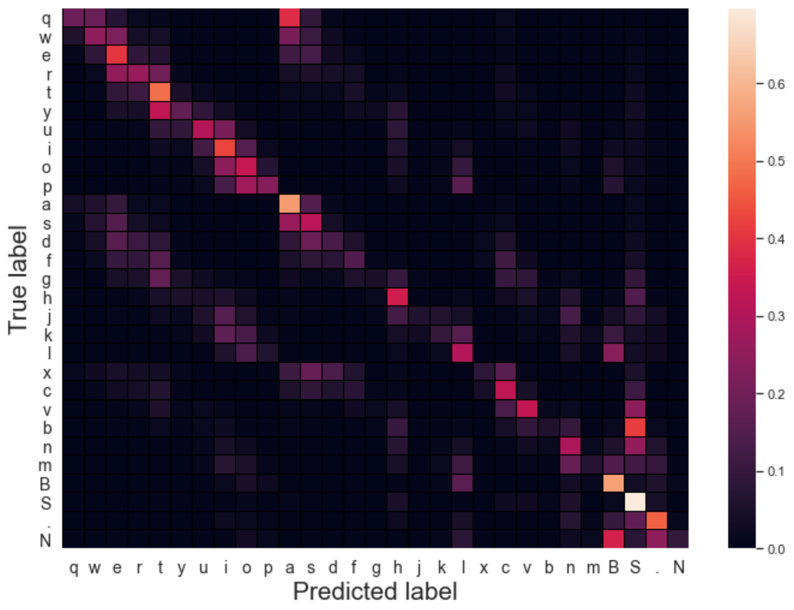


Figure 15: Heatmap describing the distribution of predicted labels for each true label in the general problem, in the general case. The cells are normalized by each true label, and the colormap is **not truncated**. The last characters in the list are 'B': Backspace, 'S': Space, '.': Period, and 'N': New line/Enter.

B Subpar model structures

Before deciding on using the models described in Sections 3.2 and 3.3, many other model details were tried but were ultimately abandoned, as they performed worse than the alternative. These include, but are not limited to:

- Using a masking [27] layer at the beginning of the model, letting the model know not to update the weights for time-steps where all values of the input are 0.
- Using the motion sensor data from Δt ms before to Δt ms after the key down event as input to the models, instead of using Δt ms before the key down event to Δt ms after the key *up* event. This removed the problem of samples being of differing length. Different values of Δt were tried.
- Using the hold times and the two RP-latencies (see Figure 6) surrounding a keystroke as additional features. These were concatenated to the output of the LSTM layers, as input to the dense layers.
- Normalising the inputs to unit variance and zero mean. This is probably best practice, and did only perform slightly worse than not doing it, but as the all inputs were of similar magnitude it was not necessary. When using the additional features described above, which could be of one or two orders of magnitude larger than the motion sensor measurements, normalising helped.
- Grouping inputs by length, creating batches where all samples are of the same length, to remove the need for padding the inputs. This did not notably affect model performance, but increased the training time somewhat.
- Not using Batch Normalization layers.
- Many different versions of different number of layers, nodes in each layer, activation functions and loss function.
- In the general problem, using different class weights.

Master's Theses in Mathematical Sciences 2021:E68
ISSN 1404-6342

LUTFMS-3433-2021

Mathematical Statistics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lu.se/