Institute for Visualization and Interactive Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelor Thesis

Generalizable Encoding for Keyboard and Mouse Data

Manuel Schwartz

Course of Study:	Softwaretechnik
Examiner:	Prof. Dr. Andreas Bulling
Supervisor:	M.Sc. Guanhua Zhang
Commenced:	November 28, 2022
Completed:	May 28, 2023

Abstract

Applying machine learning to keyboard and mouse data is an important topic in human-computer interaction since gained knowledge from analyzing user interaction behaviour allows to improve system attributes such as interactivity and user experience. For this purpose, an expressive data representation is crucial for achieving meaningful predictive power. In contrast to previous works which mostly rely on handcrafted features, this work explores generalizable encodings in order to supply the machine learning model with less prefiltered inputs.

Results on two datasets show that the proposed encodings can improve performance of interactive task recognition, since a time series representation, keeping track of mouse pointer coordinates and mouse button states in fixed time intervals, significantly outperformed the baseline of using handcrafted features in case of mouse data. Regarding keyboard data, applying a similar representation which tracks the key states also resulted in better predictive power than using manually extracted features. In addition, approaches based on techniques from natural language processing were competitive to the time series representation. This indicates that multiple encodings need to be considered when assessing how to encode keyboard data.

Overall, our work shows that applications based on machine learning on keyboard and mouse data can benefit from selecting a less prefiltering encoding technique over handcrafted feature extraction.

Contents

List of Figures

List of Tables

Acronyms

CNN convolutional neural network. 16

GRU gated recurrent unit. 17

HCI human-computer interaction. 13

ML machine learning. 13

MLP multilayer perceptron. 29

NLP natural language processing. 16

RNN recurrent neural network. 16

SERP search engine results page. 16

UI user interface. 13

ViT Vision Transformer. 27

1 Introduction

Recently, machine learning (ML) on keyboard and mouse interactions has gained importance due to the availability of this type of data on many devices [XSB16; ZBH+23] as well as the interesting applications of modelling interactive behaviour [AFB21]. For example, knowing which task the user currently works on is beneficial for interactive systems and therefore crucial for adaptive user interfaces (UIs) [FKW+17; HBLW21; ZBH+23].

Currently, predictions based on these modalities are usually performed by traditional ML models on tabular data after manually extracting features like duration of or latency between keystrokes for keyboard data and click duration or speed of movement for mouse data as well as more sophisticated ones [AL20; SBSB18]. However, given the feasibility of encoding for example text and image data such that it can be used as input to classifiers without the need for manual feature extraction [CGJ18; KRS21], it is promising to adapt such encoding methods to keyboard and mouse data, especially because of their similarities with natural language and images respectively.

Using a generalizable encoding as an alternative to manual feature extraction offers huge possibilities such as saving development time as well as making ML on keyboard and mouse data more accessible to data scientists being less experienced in the domain of human-computer interaction (HCI) [AL20]. Further, outsourcing feature extraction from the data scientist to the ML model itself could improve performance since important information might not be considered and therefore be dropped in the process of manual feature selection [NCZC21]. While it is also not guaranteed that the model extracts all the relevant features, it is not deprived of potentially useful ones a priori and can access the data in a less prefiltered manner instead of a perhaps less meaningful representation.

Thus, the contribution of our work is three-fold: (1) We propose different encoding strategies for keyboard and mouse data which are designed to maintain a high level of rawness in contrast to heavily preprocessing and prefiltering the data. (2) We compare their expressiveness and generalizability by examining their abilities to produce an output suitable as input to a classifier such that it can achieve robust performance across different datasets. (3) We provide future research with the opportunity to assess how keyboard and mouse data can be expressively encoded.

2 Related Work

Our work is related to previous works on (1) analysis of keyboard and mouse behaviour as well as (2) encoding keyboard and mouse data.

2.1 Applying Keyboard and Mouse Behaviour Analysis

A large degree of data availability makes building applications based on analyzing keyboard and mouse behaviour attractive [XSB16; ZBH+23]. Thus, there is a lot of ongoing research in this area. For example, Epp et al. [ELM11] pointed out the benefits of emotion-aware systems, being able to adapt their responses and interactions to the current needs of the user. They considered previous methods of emotion recognition like analysis of voice and facial expressions too intrusive and also impractical as special equipment is required. Thus, they developed decision trees to process features being extracted from keystroke dynamics and found that typing behaviour is also related to emotions. Based on their research, Shi et al. [SM16] conducted a study on mood-aware recommender systems. In the first stage, they developed a mood recognition tool capable of distinguishing between stressed and relaxed moods. In the second stage, they used this tool in an experiment in which the participants were shopping online in order to gain useful insights for e-commerce platforms. They found that it is worth using a mood-aware recommender system to optimize the timing for recommendations as well as to derive insights from the reaction of a user's mood to recommendations. Another application is the emotion-aware music recommender system by Jazi et al. [YKF21]. They used the relation between interaction behaviour and emotions to recommend music based on keystrokes and mouse clicks patterns. The authors reported a higher accuracy than for other music recommender systems as well as a high level of user experience.

Analyzing keyboard and mouse behaviour can also be used to identify users, which is subject to a lot of research [AFB21; LCS22; TWO+12; XSS19]. Authenticating users based on their behaviour adds another layer of security. While passwords might be guessed or get stolen and biometrics like fingerprints or facial recognition require suitable equipment and also do not prevent from someone getting access to an already logged in session, it is hard to imitate someone's interaction behaviour and it can be confirmed continuously without additional hardware [LCS22; XSS19]. Similarly, Niu et al. also successfully distinguished bots from real users by applying machine learning methods to mouse dynamics [NCZC21].

Another downstream task when analyzing keyboard and mouse behaviour is interactive task recognition. Knowing which task the user is working on is crucial for intelligent interactive systems in order to adapt to the user's intents and interaction goals [FKW+17; HBLW21; ZBH+23]. While previous research mostly relies on eye movements [HBLW21], leveraging predictions based on keyboard and mouse interactions is beneficial in terms of data availability [ZBH+23] without the need for additional hardware. Not only did Fu et al. show that this is feasible [FKW+17], but Zhang et al.

even found that this can increase predictive power [ZHL+22]. Based on this, Zhang et al. explored the similarities of interaction behaviour and natural language, both having sequential as well as hierarchical structure, and successfully predicted interactive tasks by encoding mouse and keyboard data with methods originally known from natural language processing (NLP) [ZBH+23].

2.2 Encoding Keyboard and Mouse Data

While previous work on processing keyboard and mouse data mainly focused on extracting handcrafted features like duration of or latency between keystrokes for keyboard data and click duration or speed of movement for mouse data as well as more sophisticated ones [AL20; SBSB18], there is also existing research on encoding mouse and keyboard data in a different way.

To investigate replacing the common but time-consuming and expertise-requiring process of manual feature extraction for tracking mouse cursor movements, Arapakis et al. tested and compared different encodings for predicting user attention [AL20]. They captured the mouse movements of users interacting with a search engine results page (SERP) to predict whether the user paid attention to a displayed ad. It has been shown that convolutional neural networks (CNNs) receiving heatmaps or various visualizations of trajectories outperform recurrent neural networks (RNNs) given as input a time series representation of the x and y coordinates of the mouse pointer. Though their work provides useful insights into this research area, their results are limited in terms of comparability and generalizability. Neither did they use a setting in which they could have compared with results from the literature as they collected and used their own dataset, nor did they test the performance of using handcrafted features on it themselves. Further, the encodings were evaluated in just a single dataset, containing data of always the same laboratory task of relatively short duration. Their encoding method also does not consider mouse clicks but just mouse movements.

Concerning keyboard data, Wampfler et al. pursued a similar idea by predicting affective states based on the typing behaviour of smartphone users, as they used a CNN architecture that received heatmaps of keystrokes as input [WKS+22]. A single input instance consisted of three heatmaps, each representing a differently measured time interval between two consecutive keystrokes. For each combination of two consecutive keys, there was an entry in the two-dimensional heatmap being darker the longer the average time interval between these keys was. They compared this approach to a slightly worse performing one receiving heatmaps of smartphone sensor data and a slightly better performing one receiving both input types. Although the results were quite reasonable, they lack comparability with conventional approaches as well as generalizability to other use cases. Just like Arapakis et al. [AL20], they collected and used their own dataset without any state of the art results in literature. They also did not compare to common methods for processing keyboard data, but to predictions based on sensor data, a different modality for affective state prediction. Further, data preprocessing already preselected certain features by considering only three types of averaged time intervals for two consecutive keystrokes instead of encoding the data in a less prefiltered manner. Moreover, even though their encoding method should be applicable to conventional computer keyboards as well, they only evaluated its expressiveness on data collected on touch keyboards of smartphones.

Xiaofeng et al. also worked on representations of keyboard data [XSS19]. They encoded a keystroke sequence as a sequence of vectors where each vector covers the keys, dwell times and release-press as well as press-press flight times of each pair of two consecutive keystrokes in the sequence. The

output was processed by an architecture combining a CNN with a gated recurrent unit (GRU) network suitable for sequential data. In contrast to the aforementioned research, they evaluated their approach on a well-known dataset and also compared to results from the literature. However, only one single data representation was assessed which did not manage to outperform the baseline of using even relatively simple handcrafted features.

Thus, the main novelty of this work is to compare various encoding strategies in a broad and standardized benchmark. Also, unlike heavily preprocessing the data, the proposed techniques are designed to maintain a high level of rawness, allowing the ML model to extract meaningful features autonomously instead of receiving a potentially less meaningful representation.

3 Approach

Figure 3.1 shows the pipeline of our method. The data is processed by the encoding steps marked in blue such that it can be passed to a classifier outputting the prediction. The encoding procedure is the focus of this work and is split up into two parts.

While first level encoding converts the raw data to the actual input data by applying a fixed algorithm, second level encoding, which is part of the model and consists of trainable parameters, receives this interim result and further encodes it to an input being compatible to the classifier. This is necessary because the same classifier is used for all encoding methods in order to have a meaningful comparison. Since a Transformer-based classifier is used (see Section 3.3) which expects its inputs to be sequences of tokens, second level encoding can be regarded a tokenizer.

3.1 Datasets and Preprocessing

In order to evaluate the encoding strategies, task recognition is chosen as an exemplary application area since it is of interest due to its contribution to the improvement of adaptive UIs of interactive systems [FKW+17; HBLW21; ZBH+23]. The downstream task is performed on two different datasets.

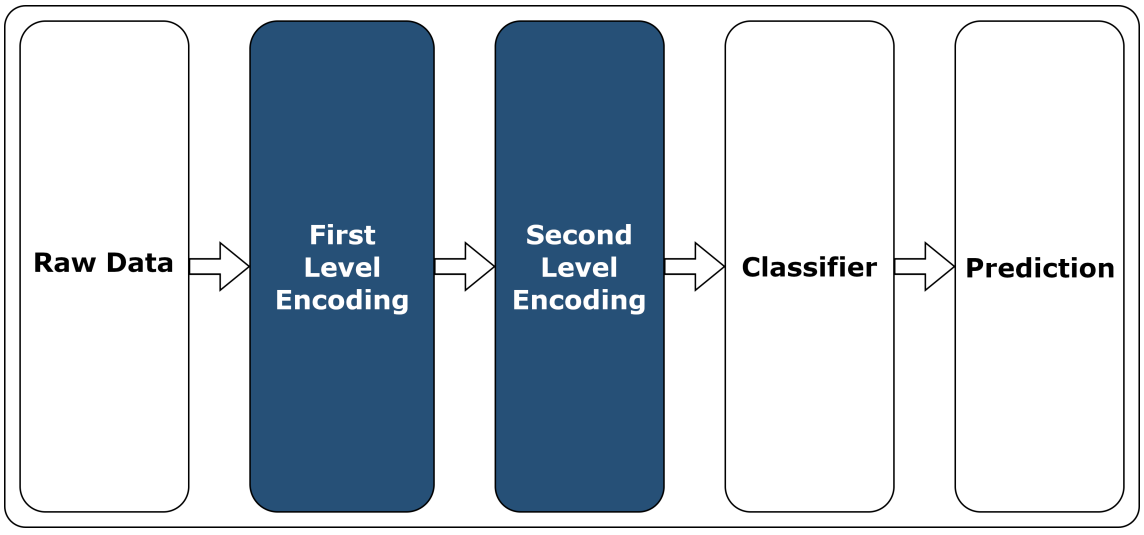

Figure 3.1: From raw data to prediction – the pipeline used in this work.

3.1.1 Buffalo Dataset

The “Shared keystroke dataset for continuous authentication” by Sun et al. [SCU16], also referred to as “Buffalo dataset”, offers about 5.5 M mouse actions and 2.5 M keystrokes from 148 participants collected in a laboratory setting. Besides transcription of a pre-defined text, the participants also performed the task of free text routine work like replying to emails or answering questions.

A window size of 10 seconds with an overlap of 5 seconds is used to split the data into individual instances. For mouse data, this results in 42, 902 instances of 156 actions each and a class distribution of 54 % to 46 %. For keyboard data, following the work of others [SCU16; XSS19; ZBH+23], only the so-called baseline data of 75 user was used, resulting in 91, 143 instances of 54 actions each and a class distribution of 60 % to 40 %.

3.1.2 EMAKI Dataset

The “Everyday Mouse And Keyboard Interactions” (EMAKI) dataset by Zhang et al. [ZBH+23] contains about 1.2 M mouse actions and 210 K keystrokes from 39 participants collected in an in-the-wild online study, meaning that the participants were using their own computers. They performed three different tasks which were text entry and editing, image editing as well as questionnaire completion.

Due to the smaller amount of data, a window size of 5 seconds with an overlap of 4 seconds is used. For mouse data, this results in 42, 410 instances of 116 actions each and a class distribution of 45 % to 31 % to 24 %. For keyboard data, this leads to 28, 420 instances of 25 actions each and a class distribution of 88 % to 6 % to 5 %.

3.2 Encodings

Mouse and keyboard data are treated separately, meaning that predictions are made considering only one of both modalities. This is due to their different characteristics, keyboard data being discrete and mouse movements being continuous. According to that, the proposed encoding techniques are designed to encode only one modality and can therefore be grouped into mouse encodings and keyboard encodings. However, there are similarities between the two groups. Moreover, second level encoding does not depend on the modality but on the structure of the output of the selected first level encoding. Therefore, most of the second level encodings are relevant for both modalities.

Every encoding method which produces classifier inputs of variable size also pads or truncates the data to the same length. This is done by setting the sequence length to the third quartile of the individual sequence lengths occurring in the training data, resulting in 75 % of the instances being padded and 25 % being truncated. Hence, a compromise between losing a lot of data due to truncation and ending up with padding-dominated data due to outliers with a high sequence length was made.

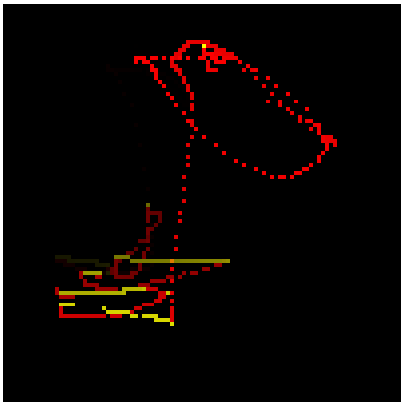

Figure 3.2: Example for mouse encoding *Trajectories*. Red (first color channel active) lines capture mouse movements in general, yellow (first and second channel active) dots correspond to clicks and yellow lines indicate drag-and-drop events. In this example, there are no purple (first and third channel active) and white (all three channels active) pixels because the right mouse button was neither pressed alone nor together with the left one.

3.2.1 First Level Encoding for Mouse Data

Due to the similarity of mouse motion data and images, the approach of Arapakis et al. [AL20] of representing mouse data as images in order to apply an image recognition model is picked up. However, mouse trajectories usually differ more from each other than in their work, since all users had to perform the same quite precisely defined task there. With the goal of increasing generalizability, this representation is adapted in how it encodes time information and is extended to additionally encode information about click events in order to cover all the data produced by logging mouse usage. Using just a single image for each data instance is also compared to using a sequence of images since this injects time information on a different level. As mouse data inherently is of sequential character, treating it as a time series in order to perform multivariate time series classification is also compared to generating images and applying image recognition. Additionally, an encoding strategy which follows the common approach of manual feature extraction is used as a baseline.

Trajectories An image representation of the mouse motion based on "Trajectories with variable line thickness" as proposed by Arapakis et al. [AL20] in section 4.2 (4). The mouse movement is encoded into an image by tracing the mouse trajectory and coloring the pixels passed while setting all other pixels to black. In contrast to their work, instead of varying line thickness to encode time information, pixel brightness is increased linearly from 0 % to 100 % as time progresses. Further, the usual three channels of an RGB image are used to encode click information by capturing all kinds of mouse movements in the first channel and activating the second or third channel additionally while the left or right mouse button is pressed, respectively. Hyperparameters are image resolution which was fixed to 128×128 pixels and if to apply linear increase or if to use constant brightness for which we compare both. A visualization of an example instance with activated change of brightness can be seen in Figure 3.2.

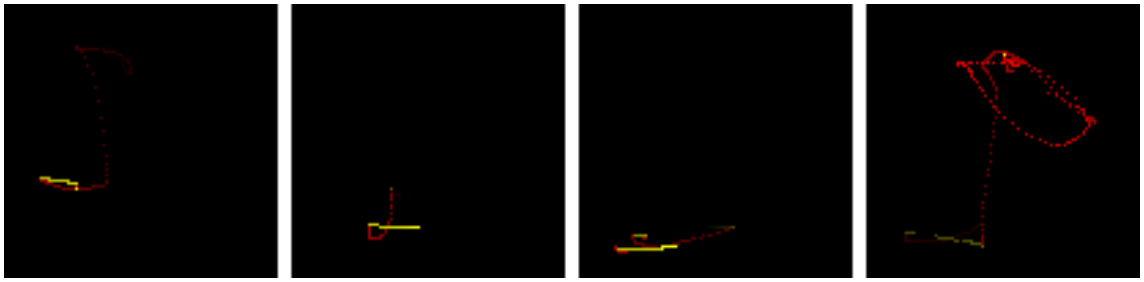

Figure 3.3: Example for mouse encoding *Sequence of Sub-Trajectories* with 4 images. Note how the darkest (earliest) pixel's position corresponds to the brightest (latest) pixel's position of the previous image since at this time the transition from one image to the next takes place.

Δt (ms)	x	y	left button	right button
...	...	...	...	...
8	0.3242	0.5649	0	0
8	0.3260	0.5649	0	0
231	0.3260	0.5649	1	0
143	0.3260	0.5649	1	0
1	0.3260	0.5649	0	0
88	0.3260	0.5667	0	0
...	...	...	...	...

Table 3.1: Excerpt of an example for mouse encoding *Time Series*. Note that Δt (ms) denotes the time difference to the previous row, also referred to as interaction gap, and is standardized based on the training data. However, non-standardized values are shown in this visualization to highlight the underlying meaning.

Sequence of Sub-Trajectories Analogous to the previous encoding, but one single data instance consists of a sequence of images instead of just a single image in order to inject time information on a different level as well as to avoid too much fading of movements at the beginning of the interaction window. For this, the interaction window is split into several sub-windows of equal duration and the *Trajectories* encoding is applied to each sub-window individually. In addition to the prior hyperparameters, which were fixed to an image resolution of still 128×128 pixels and to apply linear change of brightness, the number of images has to be chosen as well. For the experiments within the scope of this work, it was fixed to 5 since it divides both window sizes used. Figure 3.3 shows a visualization of the previous example instance for a number of images of 4.

Time Series A multivariate time series where each row representing a time step created by a user interaction contains the following values:

- The standardized time difference to the previous event, meaning that a value of 0 represents the average gap between two interactions based on the training data,
- The current x-coordinate of the mouse pointer,

- The current y-coordinate of the mouse pointer,
- A boolean value representing the current state of the left mouse button,
- A boolean value representing the current state of the right mouse button.

Coordinates are scaled to a range of $[-1, 1]$ such that a value of 0 corresponds to the middle of the screen while -1 and 1 refer to the left and right (x-coordinate) as well as the upper and lower (y-coordinate) edge. There are no hyperparameters related to this encoding. An excerpt of an example for this encoding is given in Table 3.1.

Synchronized Time Series Analogous to the previous encoding but without the first column, since instead of using time difference to the previous event as an additional feature, the mouse state is resampled at a fixed frequency. Therefore, unlike the once event-driven time series, there is a fixed time step between each row and its subsequent one. This encoding introduces resampling frequency as a hyperparameter, for which we compare 20 Hz and 50 Hz.

Handcrafted Features In order to compare the different encoding methods to the common approach of manual feature extraction, representing a data instance by a vector of handcrafted features is used as a baseline. For this purpose, we extract the features used by Traore et al. [TWO+12] to process mouse data, most of which can also be found in the overview of commonly used features collected by Salmeron-Majadas et al. [SBSB18]. As in their work, the direction of a movement is mapped onto one out of eight areas of equal size. Concretely, we use:

- *Average click time:* The average duration from pressing to releasing the mouse button,
- *Silence ratio:* The percentage of mouse events without a change of cursor position,
- *Percentage of mouse action per mouse movement direction* (8 features): The percentage of mouse events representing a cursor movement in the corresponding direction,
- *Percentage of distance per mouse movement direction* (8 features): The percentage of distance travelled with the cursor in the corresponding direction,
- *Percentage of mouse move time per mouse movement direction* (8 features): The time proportion the cursor was moved in the corresponding direction,
- *Average distance per mouse movement direction* (8 features): The average change of cursor position per mouse event for each direction,
- *Average velocity per mouse movement direction* (8 features): The average velocity of cursor movements in the corresponding direction,
- *Average velocity in x-axis per mouse movement direction* (8 features): The average velocity along the x-component of the decomposed velocity of cursor movements in the corresponding direction,
- *Average velocity in y-axis per mouse movement direction* (8 features): The average velocity along the y-component of the decomposed velocity of cursor movements in the corresponding direction,

- *Equally weighted average velocity per mouse movement direction* (8 features): The average velocity of cursor movements in the corresponding direction, but equally weighted after measuring it for each move event individually instead of calculating the overall average velocity by dividing total distance by total time.

Note that the last 8 features are constructed as a replacement for the original features "Average tangential velocity per mouse movement direction" used in the paper [TWO+12] because the latter cannot be used in general. Tangential velocity describes the component along the actual path the user plans to perform of the decomposed velocity of a cursor movement, but this path is usually unknown. This encoding does not introduce any hyperparameters.

3.2.2 First Level Encoding for Keyboard Data

Unlike mouse data, keyboard data is more similar to natural language than images. Therefore, this work follows up on the approach of Zhang et al. [ZBH+23] of treating keyboard data as natural language by processing it with different NLP tokenizers. However, an image representation is also part of the comparison as other researchers like Wampfler et al. already achieved reasonable results in previous work by generating images from keyboard data [WKS+22]. In contrast to their work, in which heatmaps of keystroke patterns were used, considering consecutive pairs of keystrokes only and already averaging over multiple time periods is avoided in the scope of this work. Instead, the data is captured in a less preprocessed way in order to minimize prefiltering of features. Since keyboard data is also sequential in nature, a time series approach, similar to the one used for mouse data, is also applied. Finally, the common approach of manual feature extraction is employed as well, serving as a baseline again.

NLP Tokens The sequence of keyboard events is considered as text (for example "keydown Shift keydown E keyup E keyup Shift keydown Space keyup Space") which is then tokenized using an NLP tokenizer. Hyperparameters are token dimension as well as the type of tokenizer. We compare tokens of size 16 and 64 generated by BPE [SHB15] and WordPiece [WSC+16]. More precisely, `CharBPETokenizer` and `BertWordPieceTokenizer` of Hugging Face's `tokenizers` implementation [Hug] are used, realizing the original BPE and the version of WordPiece used in Google's language model BERT [DCLT18].

Keystroke Timeline Diagram In order to obtain an image representation of keyboard data, a diagram is created showing the state of each key over time. While elapsed time is on the x-axis, all keys occurring at least three times in the training data are arranged on the y-axis, plus an additional special row accumulating all other keys occurring less than three times as well as keys only occurring in the validation set. Horizontal bars at the height of the corresponding key indicate time windows in which the key is pressed. Since patch size of the corresponding second level encoding *Image Encoder* (see Section 3.2.3) needs to divide both width and height of the image, the smallest possible padding such that this requirement is fulfilled is added to the bottom and right edge. This results in an image dimension of $\left\lceil \frac{\#\texttt{keys}+1}{\texttt{patch\_size}} \right\rceil \cdot \texttt{patch\_size} \times \left\lceil \frac{\#\texttt{time\_steps}}{\texttt{patch\_size}} \right\rceil \cdot \texttt{patch\_size}$ pixels. A hyperparameter is the sampling rate for the x-axis, being set to 20 Hz and 50 Hz, which together with the window size (see Section 3.1) determines the number of time steps. An example visualization is given in Figure 3.4.

Figure 3.4: Example for keyboard encoding *Keystroke Timeline Diagram*. Each row represents the state of one key. While the beginning of a white line corresponds to the key being pressed and the end of the line to the key being released, completely black rows indicate that the key was never pressed during the encoded time window.

Δt (ms)	...	Shift	...	E	...	Space	...
...	...	...	...	...	...	...	...
212	...	1	...	0	...	0	...
117	...	1	...	1	...	0	...
32	...	1	...	0	...	0	...
56	...	0	...	0	...	0	...
343	...	0	...	0	...	1	...
29	...	0	...	0	...	0	...
...	...	...	...	...	...	...	...

Table 3.2: Excerpt of an example for keyboard encoding *Time Series*. Note that Δt (ms) denotes the time difference to the previous row, also referred to as interaction gap, and is standardized based on the training data. However, non-standardized values are shown in this visualization to highlight the underlying meaning. Further, many columns are omitted in this example for a more compact visualization.

Time Series Similar to the *Time Series* encoding for mouse data outlined in Section 3.2.1, a multivariate time series is generated. Again, each row represents one time step created by a user interaction and contains the following values:

- The standardized time difference to the previous event, meaning that a value of 0 represents the average gap between two interactions based on the training data,
- One column for each key, representing the current state of the key with a boolean value. Just like for the *Keystroke Timeline Diagram*, a key needs to occur at least three times in the training data in order to get an own column. An additional column captures the accumulation of keys being pressed less often.

Also for this encoding, there are no hyperparameters. An excerpt of an example for this encoding is given in Table 3.2.

Synchronized Time Series Analogous to the mouse encoding *Synchronized Time Series* building upon the *Time Series* representation (see Section 3.2.1), using time difference to the previous event as an additional column is replaced by resampling the key states with a fixed frequency. Again, this converts the once event-driven series into a series with a fixed time step between each row and its subsequent one. Resampling frequency arises as a hyperparameter, for which we compare 20 Hz and 50 Hz.

Handcrafted Features Just like for mouse data, representing a data instance by a vector of handcrafted features is used as a baseline in order to compare the other encoding strategies to the common approach of manual feature extraction. Again, we extract the features used by Traore et al. [TWO+12] to process keyboard data, as most of them can also be found in the collection of commonly used features published by Salmeron-Majadas et al. [SBSB18]. Specifically, we extract:

- *Mean of dwell time:* The average duration from pressing to releasing a key,
- *Mean of release-press flight time:* The average duration from releasing a key to pressing the next one,
- *Mean of press-press flight time:* The average duration from pressing a key to pressing the next one,
- *Mean of trigraph time:* The average duration for completing three subsequent keystrokes, measured from pressing the first key to releasing the third one,
- *Standard deviation of dwell time:* The standard deviation of the duration from pressing to releasing a key,
- *Standard deviation of release-press flight time:* The standard deviation of the duration from releasing a key to pressing the next one,
- *Standard deviation of press-press flight time:* The standard deviation of the duration from pressing a key to pressing the next one,
- *Standard deviation of trigraph time:* The standard deviation of the duration for completing three subsequent keystrokes, measured from pressing the first key to releasing the third one,
- *Mean of dwell time per category* (2 features): The average duration from pressing to releasing a key, measured for the categories “letter” and “other” (like “Shift”, “Space” or “Esc”) separately,
- *Percentage of occurrences per category* (2 features): The proportion of completed keystrokes for the categories “letter” and “other”,
- *Percentage of occurrences of holding multiple keys:* The percentage of keyboard states in which several keys are pressed simultaneously,
- *Average typing speed:* The number of completed keystrokes per second.

In the original paper [TWO+12] they used 4 categories being “Upper Case Keystrokes”,“Lower Case Keystrokes”, “Control Keystrokes” and “Other Keystrokes”, where “Upper Case Keystrokes” also cover symbols which require “Shift” to be pressed simultaneously or “Caps lock” to be pressed ahead. However, we only distinguished between the categories “letter” and “other”, since usually, pressing “Shift” is considered a different keystroke instead of merging these events. As a direct consequence, the two features “Mean of flight times per type of user behaviour” mentioned in the paper [TWO+12] are also dropped since these flight times refer to scenarios where exactly one of two consecutive keys belongs to the category “Upper Case Keytrokes” and where both or neither belong to this category respectively. There are no hyperparameters related to this encoding.

3.2.3 Second Level Encoding

As the outputs of the previously described encodings are quite different in terms of their structural and dimensional properties, the interim results are further encoded such that all types of data are compatible with the classifier. In particular, the data can be images, sequences of images, time series as well as sequences of NLP tokens. For each type, there exists a corresponding second level encoder designed to transform the input into a sequence of tokens since this is the structure the Transformer-based classifier expects. Although second level encoding is part of the encoding, it is also part of the model itself as it has trainable parameters being trained together with the classifier.

Sequence of Tokens Encoder This encoder is applied to sequences of NLP tokens produced by the keyboard encoding *NLP Tokens*. Only a small conversion is necessary since the data is already a sequence of tokens. However, as usual for Transformers, the NLP tokens are embedded using a PyTorch `Embedding` layer [PyTa] and positional encodings are added [VSP+17]. There are also no hyperparameters since the vocab size as well as the token dimension result from the settings of the NLP tokenizer itself.

Image Encoder This architecture is used for images generated by the mouse encoding *Trajectories* and the keyboard encoding *Keystroke Timeline Diagram*. Just like in the Vision Transformer (ViT) [DBK+20], each image is divided into patches which are then flattened, transformed linearly and appended to a learnable class token. Since positions of the patches in the image are not absolute, learnable positional encodings are added to the resulting sequence of vectors. Patch size is a hyperparameter and besides the value of 16 Dosovitskiy et al. used in the ViT for images of size 256×256, we also try patches of size 8 since the authors recommend a smaller value for a lower image resolution [DBK+20].

Video Encoder Sequences of images, occurring in the context of the mouse encoding *Sequence of Sub-Trajectories*, are processed using the *Video Encoder*. While the *Image Encoder* transformed an image into a sequence of tokens, each image has to be converted to only one token when dealing with sequences of images. In order to obtain a compact vector representation of an image, the common approach is to use a CNN architecture [CGJ18]. Concretely, a two-dimensional convolution followed by a two-dimensional max pooling is applied two times before flattening the matrix to a vector and applying a linear transformation. Both convolutions as well as the linear transformation are followed by a ReLU activation. After encoding each image of the sequence, just as for the last step

of the *Sequence of Tokens Encoder*, fixed positional encodings [VSP+17] are added to the resulting sequence of vectors since the order of images in a sequence of image is well-defined. Besides the token dimension specified by the amount of neurons in the linear transformation, kernel size of the convolution layers as well as the pooling layers are also hyperparameters. We chose 64 for the first one while 3 and 2 are used for the other two since these are common values in literature [CGJ18].

Multivariate Time Series Encoder This second level encoding is designed to handle multivariate time series produced by the keyboard and mouse encodings *Time Series* and *Synchronized Time Series*. As time series is already a sequence, the set of variables of each time step is transformed into one token. Instead of directly using each row of the series as one token as is, a row-wise linear transformation followed by a ReLU activation is applied in order to inject spatial dependencies between the features. Fixed positional encodings [VSP+17] are added to the resulting sequence of vectors. The number of output features produced by the linear transformation is a hyperparameter. For mouse data, using the number of incoming features (5 for *Time Series* and 4 for *Synchronized Time Series*) is compared to using a larger size of 32 to have more room to combine features. For keyboard data where the number of features is way larger and also depends on the training data (number of unique keys in the training set plus one for the accumulation of unknown keys plus one for the time difference in case of *Time Series*), we use 16 and 64.

Handcrafted Features Encoder This encoder is used along with manually extracted features generated by the *Handcrafted Features* encodings of both mouse and keyboard data. The feature vector to be processed has no sequential characteristics. Hence, yet a sequence is expected by the classifier, it is interpreted as a sequence of length 1. Just as in the *Multivariate Time Series Encoder*, instead of directly using the feature vector as the token, a linear transformation followed by a ReLU activation is applied in order to inject spatial dependencies between the features. Again, the number of output features produced by the linear transformation is a hyperparameter. For mouse data, using the number of incoming features (66) is compared to using a smaller size of 16. For keyboard data, using the number of incoming features (14) is compared to using a larger size of 64.

3.3 Classifier

In order to obtain a meaningful comparison, the same classifier is used for all encoding methods. Transformers were originally introduced in the field of NLP [VSP+17], but also show impressive results in other areas [XPS+22]. Especially, they are also applicable to all types of data produced by the first level encoding methods including not only NLP tokens [VSP+17], but also images [DBK+20], videos [ADH+21] and time series [LRM+21]. Besides the fact that Transformers are powerful ML models for sequential data [VSP+17], which keyboard and mouse data inherently are, these were the reasons for using a Transformer-based architecture.

In detail, the `TransformerEncoder` [PyTb] implementation of PyTorch is used, consisting of two `TransformerEncoderLayers` [PyTc]. In comparison to the default parameters of $2,048$ and 0.1, both layers use a rather small feedforward network model of size 128 and a relatively high dropout value of 0.2. This decision was made to fight overfitting since the the data amount is significantly smaller than in usual NLP tasks Transformers were originally designed for. Mean pooling is applied to the

output of the `TransformerEncoder` to hand it to a multilayer perceptron (MLP). The MLP consists of a 32 neuron input layer with a dropout rate of 0.4, whereas the dimension of the output layer depends on the task and corresponds to the number of classes as usual. ReLU is used as activation function for both the `TransformerEncoder` as well as the MLP.

There would also be the option to tune the mentioned hyperparameters. However, as the focus of this work is on the encoding procedure, a reasonable comparison of performance should be possible without exhaustively optimizing the classifier itself. Since there are already a lot of encoding-related hyperparameters, the classifier-related ones are therefore kept constant.

Nevertheless, model dimension as well as the number of heads in the multi-head attention model, both hyperparameters of the `TransformerEncoderLayers`, depend on the selected encoding method, since, due to the way a Transformer-based architecture works, the former needs to correspond to the token dimension and must also be divisible by the latter. While token dimension is given by the first and second level encoding and their settings, a suitable value for the number of heads needs to be found. As Vaswani et al. employed 8 heads [VSP+17] and most of the token dimensions are a multiple of 8, this value is used as long as applicable as well. For token dimensions not being divisible by 8, the nearest possible value to 8 is used. In detail, this concerns the mouse encodings *Time Series* and *Synchronized Time Series* along with the number of output features of the *Multivariate Time Series Encoder* being set to the number of input features, which results in token dimensions and therefore also the number of heads of 5 and 4 respectively. In addition, the *Handcrafted Features* encodings of both keyboard and mouse data used in combination with the number of output features of the *Extracted Features Encoder* being set to the number of input features are also affected, leading to token dimensions of 14 and 66 and therefore a number of heads of 7 and 6 respectively.

3.4 Training and Evaluation

To train the model, which consists of second level encoding as well as the `TransformerEncoder` and MLP that form the classifier, 5-fold cross validation is used. The split is done between users, meaning that each fold contains all the data of one fifth of the participants.

Adam Optimizer with a weight decay of 10^{-4} to add L2 regularization is used to adjust the model parameters based on Cross-Entropy loss. This is done for 100 epochs with a batch size of 128, using a learning rate of 10^{-3} in one run and 10^{-4} in a second run. The comparison between the encoding methods is based on the better performance of the two runs.

Performance is measured by validation accuracy since it is very intuitive and also expressive when dealing with balanced datasets. However, expressiveness decreases with increasing class imbalance. Hence, macro F_1 score on the validation set is used in this case. Specifically, this applies to task recognition on the EMAKI keyboard dataset with 3 classes and a proportion of the majority class of 88 %.

4 Experiments

As outlined in Section 3.2, the encoding techniques presented therein encode either mouse or keyboard data due to the different characteristics of both. Hence, evaluation of the experiments described in Section 3.1 is also done separately. The significance of differences in performance was tested using a one-tailed Wilcoxon signed-rank test [SBH21]. A significance level of $\alpha = 0.05$ was chosen, while using $\alpha = 0.01$ would always result in a negative test due to the relatively small sample size of 5 when comparing two results of a 5-fold cross validation. Specifically, this results in a critical value of $W_c = 0$, meaning that each fold of the assumed better configuration has to show a higher validation metric than the according fold of the assumed worse configuration such that the difference is accepted to be significant.

4.1 Task Recognition Based on Mouse Data

The best performing hyperparameter configuration for each mouse encoding is illustrated in Table 4.1 for the EMAKI dataset and in Table 4.2 for the Buffalo dataset. Results of all hyperparameter configurations can be found in the appendix in Table A.1 and Table A.2, respectively. While *Synchronized Time Series* encoding performs best, reaching a validation accuracy of 0.7448 for

Encoding	Hyperparameters	# Params	Val. Accuracy
Synchronized Time Series	Resampling frequency: 50 Hz Output features: 32 Learning rate: 10^{-4}	26,723	0.7448 ± 0.0309
Sequence of Sub-Trajectories	Learning rate: 10^{-3}	242,071	0.6489 ± 0.0171
Trajectories	Change brightness: False Patch size: 8 Learning rate: 10^{-3}	97,954	0.6483 ± 0.0224
Handcrafted Features	Output features: 66 Learning rate: 10^{-3}	76,749	0.5959 ± 0.0234
Time Series	Output features: 5 Learning rate: 10^{-3}	3,427	0.3746 ± 0.0589

Table 4.1: Mean and standard deviation (separated by $\pm$) of accuracy for task recognition on EMAKI mouse data. Results are sorted by performance and only the best set of hyperparameters is shown for each encoding. There are three classes and the portion of the majority class is 44.52 %.

Encoding	Hyperparameters	# Params	Val. Accuracy
Synchronized Time Series	Resampling frequency: 50 Hz Output features: 32 Learning rate: 10^{-4}	26,690	0.6397 ± 0.0158
Handcrafted Features	Output features: 16 Learning rate: 10^{-3}	12,466	0.6078 ± 0.0111
Trajectories	Change brightness: False Patch size: 8 Learning rate: 10^{-4}	97,954	0.5757 ± 0.0230
Sequence of Sub-Trajectories	Learning rate: 10^{-4}	242,038	0.5431 ± 0.0439
Time Series	Output features: 5 Learning rate: 10^{-4}	3,394	0.4888 ± 0.0547

Table 4.2: Mean and standard deviation (separated by $\pm$) of accuracy for task recognition on Buffalo mouse data. Results are sorted by performance and only the best set of hyperparameters is shown for each encoding. There are two classes and the portion of the majority class is 54.02 %.

EMAKI and 0.6397 for Buffalo dataset, *Time Series* encoding achieves by far the worst results, even worse than always predicting the majority class, for both datasets. Analyzing the outperformance of *Synchronized Time Series* encoding shows that it is significant, since comparing it to the second best encoding, being *Sequence of Sub-Trajectories* in case of EMAKI and *Handcrafted Features* encoding in case of Buffalo dataset, yields a calculated test statistic of $W = 0 \leq W_c$ both times. Further, the same result is obtained when comparing it to *Handcrafted Features* encoding on EMAKI, confirming that it beats the baseline significantly on both datasets. Using *Synchronized Time Series* encoding with a resampling frequency of 50 Hz and 32 output features in the linear layer of the associated second level encoding *Multivariate Time Series Encoder* seems to be the best setting in both cases. However, this observation is not statistically significant. Comparing this configuration to 20 Hz and 32 output features, which achieved the second highest accuracy on both datasets, results in $W = 2$ for EMAKI and $W = 7$ for Buffalo, both exceeding the critical value of $W_c = 0$. Whereas for EMAKI, all encodings except *Time Series* beat the baseline of applying *Handcrafted Features* encoding, for Buffalo, only *Synchronized Time Series* encoding outperforms the baseline.

4.2 Task Recognition Based on Keyboard Data

The best performing hyperparameter configuration for each keyboard encoding is presented in Table 4.3 for the EMAKI dataset and in Table 4.4 for the Buffalo dataset. Results of all hyperparameter configurations can be found in the appendix in Table A.3 and Table A.4, respectively. Every encoding outperforms the baseline of applying *Handcrafted Features* encoding significantly. This is shown by performing a Wilcoxon test for each encoding strategy but *Handcrafted Features* encoding between this strategy and the baseline for both datasets, resulting in a calculated test statistic of $W = 0 \leq W_c$ in all 8 cases. While encoding the instances as a *Keystroke Timeline Diagram* seems

Encoding	Hyperparameters	# Params	Val. Macro F_1
Synchronized Time Series	Resampling frequency: 20 Hz Output features: 64 Learning rate: 10^{-4}	74,499	0.5302 ± 0.0559
NLP Tokens	Tokenizer: `BertWordPiece` Output features: 16 Learning rate: 10^{-3}	12,563	0.5246 ± 0.0661
Time Series	Output features: 64 Learning rate: 10^{-4}	74,563	0.5219 ± 0.0272
Keystroke Timeline Diagram	Resampling frequency: 50 Hz Patch size: 8 Learning rate: 10^{-4}	94,531	0.4968 ± 0.0525
Handcrafted Features	Output features: 64 Learning rate: 10^{-3}	70,083	0.4213 ± 0.0336

Table 4.3: Mean and standard deviation (separated by $\pm$) of macro F_1 score for task recognition on EMAKI keyboard data. Results are sorted by performance and only the best set of hyperparameters is shown for each encoding. There are three classes and the portion of the majority class is 88.31 %. Always predicting it would yield a macro F_1 score of 0.3126, whereas answering at random would result in 0.3333.

Encoding	Hyperparameters	# Params	Val. Accuracy
NLP Tokens	Tokenizer: `BertWordPiece` Output features: 64 Learning rate: 10^{-3}	73,954	0.8301 ± 0.0167
Time Series	Output features: 64 Learning rate: 10^{-3}	74,978	0.7826 ± 0.0178
Synchronized Time Series	Resampling frequency: 50 Hz Output features: 64 Learning rate: 10^{-4}	74,914	0.7511 ± 0.0201
Keystroke Timeline Diagram	Resampling frequency: 20 Hz Patch size: 8 Learning rate: 10^{-4}	92,578	0.7226 ± 0.0074
Handcrafted Features	Output features: 14 Learning rate: 10^{-3}	10,000	0.6699 ± 0.0170

Table 4.4: Mean and standard deviation (separated by $\pm$) of accuracy for task recognition on Buffalo keyboard data. Results are sorted by performance and only the best set of hyperparameters is shown for each encoding. There are two classes and the portion of the majority class is 60.11 %.

to result in the smallest improvement for both datasets, this presumption is not of significance. In case of EMAKI, comparison to *Time Series* encoding yields a test statistic of $W = 2$ while a value of $W = 1$ is obtained when comparing to *Synchronize Time Series* on Buffalo. The ranking of the other techniques depends on the dataset. For EMAKI, *Synchronized Time Series* encoding achieves the best macro F_1 score of 0.5302 followed by *NLP Tokens* (0.5246) and *Time Series* (0.5219) encoding. However, this slight outperformance is not significant either since comparison to the *NLP Tokens* encoding results in a calculated test statistic of $W = 6$, clearly exceeding the critical value of $W_c = 0$. For Buffalo, the highest accuracy of 0.8301 is accomplished by *NLP Tokens*, followed by *Time Series* (0.7826) and *Synchronized Time Series* (0.7511). For every other encoding strategy, performing the Wilcoxon test between this strategy and the *NLP Tokens* encoding yields a value of $W = 0 \leq W_c$, showing that the outperformance is significant.

There is also no clear trend for which set of hyperparameters performs best. While using 64 output features instead of 16 in the linear layer of the associated second level encoding *Multivariate Time Series Encoder* seems to be beneficial for both *Time Series* and *Synchronized Time Series* on both datasets, this presumption is not of significance. When comparing each hyperparameter configuration which uses 64 features to the same but using 16 features, Wilcoxon test was positive in only half of the cases for *Synchronized Time Series* and in none of the cases for *Time Series*. Moreover, `BertWordPiece` seems to be the better choice for the NLP tokenizer in case of *NLP Tokens* encoding for both datasets. However, the slight outperformance in comparison to the `CharBPE` tokenizer is also not significant, which is shown by $W = 4$ (EMAKI) and $W = 6$ (Buffalo) for a token dimension of 16 as well as $W = 3$ (EMAKI) and $W = 5$ (Buffalo) for a token dimension of 64, all of which exceed the critical value of $W_c = 0$.

5 Discussion

From what we found in our experiments, *Synchronized Time Series* as outlined in Section 3.2 is an expressive encoding strategy in order to encode both keyboard and mouse data. It achieves robust performance and outperforms the baseline of using handcrafted features for both modalities significantly. This applies to both datasets despite their different settings. Specifically, the encoding could not only perform well on the Buffalo dataset which was collected in a laboratory setting where every participant of the baseline subset (see Section 3.1.1) used the same keyboard, but also expressively represented the instances of the in-the-wild EMAKI dataset where participants used their own hardware. As outlined in Section 3.1, they also differ in the interactive tasks the participants had to perform. Thus, the findings are consistent not only across different participants, but also across different data collection environments, interactive tasks and devices. The robust performance can be explained by the sequential structure of the behavioural data, which suggests a time series representation.

However, while *Synchronized Time Series* encoding achieves reasonable results for both modalities, it is very interesting that *Time Series* encoding is competitive for keyboard data but by far the worst in our experiments on mouse data. On the one hand, *Time Series* for mouse data as described in Section 3.2.1 and *Time Series* for keyboard data as outlined in Section 3.2.2 are two different encoding strategies, being a possible explanation for this performance gap. On the other hand, the underlying idea and structure is very similar. While *Synchronized Time Series* encoding addresses the downside of unequal time steps between user interactions in *Time Series* encoding, it introduces a considerable number of redundant observations during time periods of user inactivity. This compensation was necessary and also very effective for mouse data in our experiments since the synchronized implementation performed significantly better. However, the drawback of unequal time steps was no problem for keyboard data. A possible explanation for this is the discrete characteristics of keyboard data in contrast to the continuous ones of mouse movements, presumably requiring the interactions to be resampled at a fixed frequency only in the case of mouse data. The other potential drawback of redundant observations seemed to be no issue since *Synchronized Time Series* encoding performed well in both modalities.

Even though the outperformance is not statistically significant, a resampling frequency of 50 Hz when using *Synchronized Time Series* encoding seems to be slightly better than the lower rate of 20 Hz. A possible reason that this does not also apply to keyboard data, where sometimes 50 Hz and sometimes 20 Hz performed better, could be that using a higher frequency is particularly useful for capturing fine-grained details of mouse movements, which can thus be represented even more continuously, which they inherently are. However, as the outperformance in case of mouse data is not of significance and none of both values turned out to be better in case of keyboard data, our suggestion is to experiment with this hyperparameter when applying *Synchronized Time Series* encoding to specific use cases. This also applies to the tokenizer used in the *NLP Tokens* encoding

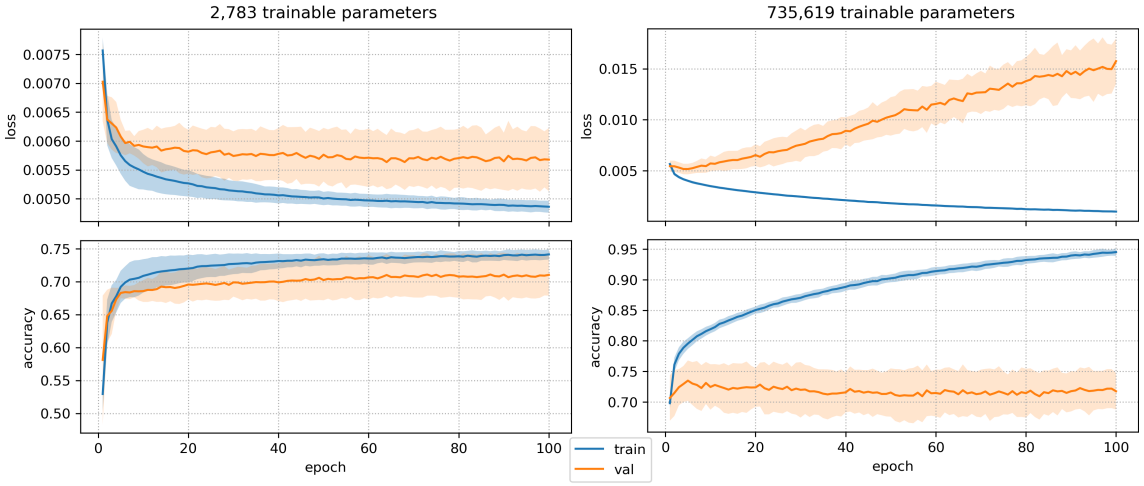

Figure 5.1: Performance comparison with different number of trainable parameters. On the left, *Synchronized Time Series* encoding with the usual second level encoding *Multivariate Time Series Encoder*, using 4 output features in its linear layer, resulting in 2,783 trainable parameters. On the right, an adapted version of the second level encoding, consisting of two linear layers of size 256, resulting in 735,619 trainable parameters. The colored area around the line denotes addition and subtraction of one standard deviation.

for keyboard data, where the slight outperformance of the `BertWordPiece` tokenizer, probably due to the generated subwords achieving a more appropriate level of granularity and contextual information, was also not significant.

Finally, we draw the conclusion that *Synchronized Time Series* encoding is clearly the best choice for encoding mouse data. While its counterpart for processing keyboard data also achieves a robust performance, comparing it to *NLP Tokens* and *Time Series* encoding might be beneficial for this modality since none of the encodings was clearly the best in this case. Our experiments have shown that using a generalizable encoding can improve predictive power over the usual approach of extracting handcrafted features and should be considered when processing keyboard and mouse data.

5.1 Influence of the Number of Trainable Encoding Parameters

Due to the different second level encodings and their output dimensions, the number of trainable parameters of the model depends on the encoding. A large number of parameters can cause overfitting which might result in a poor ability of generalization and therefore a low validation performance. Thus, the influence of this value on the performance was also inspected. For this purpose, *Synchronized Time Series* encoding with a resampling frequency of 20 Hz and 4 output features in the linear layer of the associated second level encoding *Multivariate Time Series Encoder* was referred to as a baseline. Its performance on the EMAKI dataset was compared to the same first level encoding but using two linear layers of size 256 in the second level encoding. This increases the number of trainable parameters from 2,783 to 735,619.

Figure 5.1 shows loss and accuracy during training and that the model with a larger number of parameters clearly suffers from overfitting. Training loss and accuracy are much better in comparison to the smaller model and validation loss increases after only a few epochs. However, validation accuracy decreases just slightly and ends up to be 0.7176 ± 0.0298. Compared to the smaller model's final validation accuracy of 0.7103 ± 0.0314, this indicates that the much larger number of trainable parameters did not have a negative influence on the performance. This is also consistent with the findings of Götz et al. who observed that an oversized number of trainable parameters does not significantly reduce classification performance [GGS+22]. The high validation loss but still high validation accuracy can be explained by a relatively small portion of wrong predictions being misclassified with a high level of confidence.

5.2 Limitations and Future Work

While this work provides valuable insights into encoding keyboard and mouse data for interactive task recognition, from a technical perspective, applicability of the proposed encoding methods is not limited to this downstream task only. Therefore, an interesting future work is to evaluate the encoding strategies for other tasks also. For example, transferability would offer huge chances in the field of user authentication, where users are recognized and authenticated based on their behaviour in order to achieve a higher level of security compared to using passwords only [MHHS17; SCU16; TWO+12]. In addition, if gaining user knowledge from behaviour benefits from a more expressive data representation, emotion-aware UIs to improve user-system interaction as well as mood-aware recommender systems for targeted marketing [SM16] and better user experience [YKF21] could also be improved. Since we have shown that using a generalizable encoding can outperform the usual approach of extracting handcrafted features, we invite to implement this kind of approach in real-world applications.

Future work can also explore the limits of the promising encoding techniques when further optimizing and tuning hyperparameters, especially the ones related to the classifier as well as the training routine, since these were not the focus of our experiments. Evaluating if combining keyboard and mouse data further improves performance would also be interesting. In case of applying *Synchronized Time Series* encoding to both modalities, the resulting matrices could just be concatenated such that each time step covers both sets of features. Another approach would be to process both types of data in separate networks and to connect the last hidden layers of both networks to a single final output layer. This would also enable to use encoding strategies where keyboard and mouse interactions are asynchronous and therefore unaligned as well as to even use two encodings with different types of output structures listed in Section 3.2.3.

Overall, our work offers valuable insights into encoding keyboard and mouse data for interactive task recognition and provides a solid foundation for further exploration in the context of other downstream tasks.

6 Conclusion

We explored different ways to encode keyboard and mouse data as an alternative to extract handcrafted features. Evaluated on two datasets, encoding mouse interactions as a time series of coordinates and button states with a fixed sampling rate significantly outperformed the other methods as well as the baseline in the context of task recognition. Regarding keyboard data, a similar representation of tracking the key states in a time series with fixed sampling rate was also among the best encoding methods. In contrast to mouse data, where an event-driven time series with unequal time steps between user interactions performed by far the worst, encoding keyboard data this way achieved comparable results and outperformed the baseline as well. Additionally, keyboard data was successfully treated as text describing the interaction which was then processed with NLP tokenizers. While the the ranking of these three techniques depended on the dataset, an image representation of the key states over time performed worse in both scenarios, but still outperformed the baseline of using handcrafted features.

Bibliography

[ADH+21] A. Arnab, M. Dehghani, G. Heigold, C. Sun, M. Lučić, C. Schmid. "Vivit: A video vision transformer". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 6836–6846 (cit. on p. 28).

[AFB21] M. Antal, N. Fejér, K. Buza. "SapiMouse: Mouse dynamics-based user authentication using deep feature learning". In: *2021 IEEE 15th International Symposium on Applied Computational Intelligence and Informatics (SACI)*. IEEE. 2021, pp. 61–66 (cit. on pp. 13, 15).

[AL20] I. Arapakis, L. A. Leiva. "Learning efficient representations of mouse movements to predict user attention". In: *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2020, pp. 1309–1318 (cit. on pp. 13, 16, 21).

[CGJ18] R. Chauhan, K. K. Ghanshala, R. Joshi. "Convolutional neural network (CNN) for image detection and recognition". In: *2018 First International Conference on Secure Cyber Computing and Communication (ICSCCC)*. IEEE. 2018, pp. 278–282 (cit. on pp. 13, 27, 28).

[DBK+20] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al. "An image is worth 16x16 words: Transformers for image recognition at scale". In: *arXiv preprint arXiv:2010.11929* (2020) (cit. on pp. 27, 28).

[DCLT18] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018) (cit. on p. 24).

[ELM11] C. Epp, M. Lippold, R. L. Mandryk. "Identifying emotional states using keystroke dynamics". In: *Proceedings of the sigchi conference on human factors in computing systems*. 2011, pp. 715–724 (cit. on p. 15).

[FKW+17] E. Y. Fu, T. C. Kwok, E. Y. Wu, H. V. Leong, G. Ngai, S. C. Chan. "Your mouse reveals your next activity: towards predicting user intention from mouse interaction". In: *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*. Vol. 1. IEEE. 2017, pp. 869–874 (cit. on pp. 13, 15, 19).

[GGS+22] T. I. Götz, S. Göb, S. Sawant, X. Erick, T. Wittenberg, C. Schmidkonz, A. Tomé, E. Lang, A. Ramming. "Number of necessary training examples for Neural Networks with different number of trainable parameters". In: *Journal of Pathology Informatics* 13 (2022), p. 100114 (cit. on p. 37).

[HBLW21] Z. Hu, A. Bulling, S. Li, G. Wang. "Ehtask: Recognizing user tasks from eye and head movements in immersive virtual reality". In: *IEEE Transactions on Visualization and Computer Graphics* (2021) (cit. on pp. 13, 15, 19).

[Hug] HuggingFace. *Tokenizers*. URL: https://huggingface.co/docs/tokenizers/index (visited on 05/07/2023) (cit. on p. 24).

[KRS21] K. S. Kalyan, A. Rajasekharan, S. Sangeetha. "Ammus: A survey of transformer-based pretrained models in natural language processing". In: *arXiv preprint arXiv:2108.05542* (2021) (cit. on p. 13).

[LCS22] J. Li, H.-C. Chang, M. Stamp. "Free-text keystroke dynamics for user authentication". In: *Cybersecurity for Artificial Intelligence*. Springer, 2022, pp. 357–380 (cit. on p. 15).

[LRM+21] M. Liu, S. Ren, S. Ma, J. Jiao, Y. Chen, Z. Wang, W. Song. "Gated transformer networks for multivariate time series classification". In: *arXiv preprint arXiv:2103.14438* (2021) (cit. on p. 28).

[MHHS17] C. Murphy, J. Huang, D. Hou, S. Schuckers. "Shared dataset on natural human-computer interaction to support continuous authentication research". In: *2017 IEEE International Joint Conference on Biometrics (IJCB)*. IEEE. 2017, pp. 525–530 (cit. on p. 37).

[NCZC21] H. Niu, J. Chen, Z. Zhang, Z. Cai. "Mouse Dynamics Based Bot Detection Using Sequence Learning". In: *Biometric Recognition: 15th Chinese Conference, CCBR 2021, Shanghai, China, September 10–12, 2021, Proceedings 15*. Springer. 2021, pp. 49–56 (cit. on pp. 13, 15).

[PyTa] PyTorch. *EMBEDDING*. URL: https://pytorch.org/docs/stable/generated/torch.nn.Embedding.html (visited on 05/08/2023) (cit. on p. 27).

[PyTb] PyTorch. *TRANSFORMERENCODER*. URL: https://pytorch.org/docs/stable/generated/torch.nn.TransformerEncoder.html (visited on 05/07/2023) (cit. on p. 28).

[PyTc] PyTorch. *TRANSFORMERENCODERLAYER*. URL: https://pytorch.org/docs/stable/generated/torch.nn.TransformerEncoderLayer.html (visited on 05/07/2023) (cit. on p. 28).

[SBH21] B. R. Sziklai, M. Baranyi, K. Héberger. "Testing Rankings with Cross-Validation". In: *arXiv preprint arXiv:2105.11939* (2021) (cit. on p. 31).

[SBSB18] S. Salmeron-Majadas, R. S. Baker, O. C. Santos, J. G. Boticario. "A machine learning approach to leverage individual keyboard and mouse interaction behavior from multiple users in real-world learning scenarios". In: *IEEE Access* 6 (2018), pp. 39154–39179 (cit. on pp. 13, 16, 23, 26).

[SCU16] Y. Sun, H. Ceker, S. Upadhyaya. "Shared keystroke dataset for continuous authentication". In: *2016 IEEE International Workshop on Information Forensics and Security (WIFS)*. IEEE. 2016, pp. 1–6 (cit. on pp. 20, 37).

[SHB15] R. Sennrich, B. Haddow, A. Birch. "Neural machine translation of rare words with subword units". In: *arXiv preprint arXiv:1508.07909* (2015) (cit. on p. 24).

[SM16] F. Shi, J.-L. Marini. "Can e-Commerce Recommender Systems be More Popular with Online Shoppers if they are Mood-aware?" In: *International Conference on Web Information Systems and Technologies*. Vol. 3. SCITEPRESS. 2016, pp. 173–180 (cit. on pp. 15, 37).

[TWO+12] I. Traore, I. Woungang, M. S. Obaidat, Y. Nakkabi, I. Lai. “Combining mouse and keystroke dynamics biometrics for risk-based authentication in web environments”. In: *2012 fourth international conference on digital home*. IEEE. 2012, pp. 138–145 (cit. on pp. 15, 23, 24, 26, 27, 37).

[VSP+17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, I. Polosukhin. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017) (cit. on pp. 27–29).

[WKS+22] R. Wampfler, S. Klingler, B. Solenthaler, V. R. Schinazi, M. Gross, C. Holz. “Affective State Prediction from Smartphone Touch and Sensor Data in the Wild”. In: *CHI Conference on Human Factors in Computing Systems*. 2022, pp. 1–14 (cit. on pp. 16, 24).

[WSC+16] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, et al. “Google’s neural machine translation system: Bridging the gap between human and machine translation”. In: *arXiv preprint arXiv:1609.08144* (2016) (cit. on p. 24).

[XPS+22] Z. Xia, X. Pan, S. Song, L. E. Li, G. Huang. “Vision transformer with deformable attention”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2022, pp. 4794–4803 (cit. on p. 28).

[XSB16] P. Xu, Y. Sugano, A. Bulling. “Spatio-temporal modeling and prediction of visual attention in graphical user interfaces”. In: *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. 2016, pp. 3299–3310 (cit. on pp. 13, 15).

[XSS19] L. Xiaofeng, Z. Shengfei, Y. Shengwei. “Continuous authentication by free-text keystroke based on CNN plus RNN”. In: *Procedia computer science* 147 (2019), pp. 314–318 (cit. on pp. 15, 16, 20).

[YKF21] S. Yousefian Jazi, M. Kaedi, A. Fatemi. “An emotion-aware music recommender system: bridging the user’s interaction and music recommendation”. In: *Multimedia Tools and Applications* 80.9 (2021), pp. 13559–13574 (cit. on pp. 15, 37).

[ZBH+23] G. Zhang, M. Bortoletto, Z. Hu, L. Shi, M. Bâce, A. Bulling. “Exploring Natural Language Processing Methods for Interactive Behaviour Modelling”. In: *arXiv preprint arXiv:2303.16039* (2023) (cit. on pp. 13, 15, 16, 19, 20, 24).

[ZHL+22] G. Zhang, S. Hindennach, J. Leusmann, F. Bühler, B. Steuerlein, S. Mayer, M. Bâce, A. Bulling. “Predicting Next Actions and Latent Intents during Text Formatting”. In: (2022) (cit. on p. 16).

A Results of All Hyperparameter Configurations

Encoding (Hyperparameters)	Hyperparameters	# Params	Val. Accuracy	Rank
Trajectories (change brightness, patch size, learning rate)	False, 8, 10^{-3}	97,987	**0.6483** $\pm$ 0.0224	10
	False, 8, 10^{-4}	97,987	0.6449 $\pm$ 0.0192	11
	False, 16, 10^{-3}	882,307	0.6251 $\pm$ 0.0196	15
	False, 16, 10^{-4}	882,307	0.6317 $\pm$ 0.0254	14
	True, 8, 10^{-3}	97,987	0.6339 $\pm$ 0.0118	13
	True, 8, 10^{-4}	97,987	0.6347 $\pm$ 0.0182	12
	True, 16, 10^{-3}	882,307	0.6229 $\pm$ 0.0204	16
	True, 16, 10^{-4}	882,307	0.6222 $\pm$ 0.0154	17
Sequence of Sub-Trajectories (learning rate)	10^{-3}	242,071	**0.6489** $\pm$ 0.0171	9
	10^{-4}	242,071	0.5896 $\pm$ 0.1191	20
Time Series (output features, learning rate)	5, 10^{-3}	3,427	**0.3746** $\pm$ 0.0589	23
	5, 10^{-4}	3,427	0.2974 $\pm$ 0.0649	26
	32, 10^{-3}	26,755	0.3666 $\pm$ 0.0716	24
	32, 10^{-4}	26,755	0.3218 $\pm$ 0.0934	25
Synchronized Time Series (resampling frequency, output features, learning rate)	20 Hz, 4, 10^{-3}	2,783	0.7103 $\pm$ 0.0314	4
	20 Hz, 4, 10^{-4}	2,783	0.6763 $\pm$ 0.0106	8
	20 Hz, 32, 10^{-3}	26,723	0.7029 $\pm$ 0.0379	6
	20 Hz, 32, 10^{-4}	26,723	0.7360 $\pm$ 0.0291	2
	50 Hz, 4, 10^{-3}	2,783	0.7085 $\pm$ 0.0279	5
	50 Hz, 4, 10^{-4}	2,783	0.6914 $\pm$ 0.0318	7
	50 Hz, 32, 10^{-3}	26,723	0.7130 $\pm$ 0.0270	3
	50 Hz, 32, 10^{-4}	26,723	**0.7448** $\pm$ 0.0309	**1**
Handcrafted Features (output features, learning rate)	16, 10^{-3}	12,499	0.5901 $\pm$ 0.0155	19
	16, 10^{-4}	12,499	0.5830 $\pm$ 0.0093	22
	66, 10^{-3}	76,749	0.5830 $\pm$ 0.0252	21
	66, 10^{-4}	76,749	**0.5959** $\pm$ 0.0234	18

Table A.1: Mean and standard deviation (separated by $\pm$) of accuracy for task recognition on EMAKI mouse data. Results are sorted by the encoding method and the last column indicates the ranking of the configuration relative to the others. There are two classes and the portion of the majority class is 44.52 %.

Encoding (Hyperparameters)	Hyperparameters	# Params	Val. Accuracy	Rank
Trajectories (change brightness, patch size, learning rate)	False, 8, 10^{-3}	97,954	0.5382 ± 0.0406	20
	False, 8, 10^{-4}	97,954	**0.5757** ± 0.0230	13
	False, 16, 10^{-3}	882,274	0.5702 ± 0.0116	14
	False, 16, 10^{-4}	882,274	0.5558 ± 0.0051	15
	True, 8, 10^{-3}	97,954	0.5382 ± 0.0406	21
	True, 8, 10^{-4}	97,954	0.5474 ± 0.0503	18
	True, 16, 10^{-3}	882,274	0.5478 ± 0.0358	17
	True, 16, 10^{-4}	882,274	0.5535 ± 0.0194	16
Sequence of Sub-Trajectories (learning rate)	10^{-3}	242,038	0.5382 ± 0.0406	22
	10^{-4}	242,038	**0.5431** ± 0.0439	19
Time Series (output features, learning rate)	5, 10^{-3}	3,394	0.4719 ± 0.0461	24
	5, 10^{-4}	3,394	**0.4888** ± 0.0547	23
	32, 10^{-3}	26,722	0.4680 ± 0.0464	25
	32, 10^{-4}	26,722	0.4621 ± 0.0427	26
Synchronized Time Series (resampling frequency, output features, learning rate)	20 Hz, 4, 10^{-3}	2,750	0.6171 ± 0.0142	5
	20 Hz, 4, 10^{-4}	2,750	0.6067 ± 0.0208	8
	20 Hz, 32, 10^{-3}	26,690	0.6231 ± 0.0098	4
	20 Hz, 32, 10^{-4}	26,690	0.6388 ± 0.0140	2
	50 Hz, 4, 10^{-3}	2,750	0.6108 ± 0.0160	6
	50 Hz, 4, 10^{-4}	2,750	0.5987 ± 0.0187	11
	50 Hz, 32, 10^{-3}	26,690	0.6281 ± 0.0167	3
	50 Hz, 32, 10^{-4}	26,690	**0.6397** ± 0.0158	**1**
Handcrafted Features (output features, learning rate)	16, 10^{-3}	12,466	0.6016 ± 0.0137	10
	16, 10^{-4}	12,466	**0.6078** ± 0.0111	7
	66, 10^{-3}	76,716	0.5946 ± 0.0122	12
	66, 10^{-4}	76,716	0.6051 ± 0.0090	9

Table A.2: Mean and standard deviation (separated by $\pm$) of accuracy for task recognition on Buffalo mouse data. Results are sorted by the encoding method and the last column indicates the ranking of the configuration relative to the others. There are two classes and the portion of the majority class is 54.02 %.

Encoding (Hyperparameters)	Hyperparameters	# Params	Val. Macro F_1	Rank
NLP Tokens (tokenizer, output features, learning rate)	`CharBPE`, 16, 10^{-3}	12,563	0.4929 ± 0.0456	11
	`CharBPE`, 16, 10^{-4}	12,563	0.4419 ± 0.0323	27
	`CharBPE`, 64, 10^{-3}	73,667	0.4651 ± 0.0430	22
	`CharBPE`, 64, 10^{-4}	73,667	0.4916 ± 0.0321	12
	`BertWordPiece`, 16, 10^{-3}	12,563	**0.5246** ± 0.0661	2
	`BertWordPiece`, 16, 10^{-4}	12,563	0.4534 ± 0.0362	25
	`BertWordPiece`, 64, 10^{-3}	73,667	0.5001 ± 0.0463	7
	`BertWordPiece`, 64, 10^{-4}	73,667	0.5032 ± 0.0496	6
Keystroke Timeline Diagram (resampling frequency, patch size, learning rate)	20 Hz, 8, 10^{-3}	81,859	0.4824 ± 0.0325	19
	20 Hz, 8, 10^{-4}	81,859	0.4872 ± 0.0562	16
	20 Hz, 16, 10^{-3}	742,531	0.4914 ± 0.0407	13
	20 Hz, 16, 10^{-4}	742,531	0.4892 ± 0.0392	14
	50 Hz, 8, 10^{-3}	94,531	0.4744 ± 0.0406	21
	50 Hz, 8, 10^{-4}	94,531	**0.4968** ± 0.0525	9
	50 Hz, 16, 10^{-3}	754,051	0.4865 ± 0.0459	17
	50 Hz, 16, 10^{-4}	754,051	0.4930 ± 0.0621	10
Time Series (output features, learning rate)	16, 10^{-3}	12,787	0.4985 ± 0.0617	8
	16, 10^{-4}	12,787	0.4622 ± 0.0516	23
	64, 10^{-3}	74,563	0.5138 ± 0.0486	5
	64, 10^{-4}	74,563	**0.5219** ± 0.0272	3
Synchronized Time Series (resampling frequency, output features, learning rate)	20 Hz, 16, 10^{-3}	12,771	0.4815 ± 0.0384	20
	20 Hz, 16, 10^{-4}	12,771	0.4409 ± 0.0360	28
	20 Hz, 64, 10^{-3}	74,499	0.4886 ± 0.0401	15
	20 Hz, 64, 10^{-4}	74,499	**0.5302** ± 0.0559	**1**
	50 Hz, 16, 10^{-3}	12,771	0.4578 ± 0.0548	24
	50 Hz, 16, 10^{-4}	12,771	0.4423 ± 0.0303	26
	50 Hz, 64, 10^{-3}	74,499	0.4824 ± 0.0449	18
	50 Hz, 64, 10^{-4}	74,499	0.5175 ± 0.0434	4
Handcrafted Features (output features, learning rate)	14, 10^{-3}	10,033	0.3975 ± 0.0444	31
	14, 10^{-4}	10,033	0.3867 ± 0.0338	32
	64, 10^{-3}	70,083	**0.4213** ± 0.0336	29
	64, 10^{-4}	70,083	0.4174 ± 0.0352	30

Table A.3: Mean and standard deviation (separated by ±) of macro F_1 score for task recognition on EMAKI keyboard data. Results are sorted by the encoding method and the last column indicates the ranking of the configuration relative to the others. There are three classes and the portion of the majority class is 88.31 %. Always predicting it would yield a macro F_1 score of 0.3126, whereas answering at random would result in 0.3333.

Encoding (Hyperparameters)	Hyperparameters	# Params	Val. Accuracy	Rank
NLP Tokens (tokenizer, output features, learning rate)	`CharBPE`, 16, 10^{-3}	12,610	0.7907 ± 0.0113	6
	`CharBPE`, 16, 10^{-4}	12,610	0.7509 ± 0.0050	11
	`CharBPE`, 64, 10^{-3}	73,954	0.8289 ± 0.0042	2
	`CharBPE`, 64, 10^{-4}	73,954	0.8097 ± 0.0057	3
	`BertWordPiece`, 16, 10^{-3}	12,610	0.7986 ± 0.0161	5
	`BertWordPiece`, 16, 10^{-4}	12,610	0.7441 ± 0.0170	15
	`BertWordPiece`, 64, 10^{-3}	73,954	**0.8301** ± 0.0167	**1**
	`BertWordPiece`, 64, 10^{-4}	73,954	0.8047 ± 0.0150	4
Keystroke Timeline Diagram (resampling frequency, patch size, learning rate)	20 Hz, 8, 10^{-3}	92,578	0.7078 ± 0.0036	22
	20 Hz, 8, 10^{-4}	92,578	**0.7226** ± 0.0074	19
	20 Hz, 16, 10^{-3}	753,250	0.7150 ± 0.0083	21
	20 Hz, 16, 10^{-4}	753,250	0.6805 ± 0.0079	28
	50 Hz, 8, 10^{-3} (*)	120,994	0.6956 ± 0.0093	23
	50 Hz, 8, 10^{-4} (*)	120,994	0.6856 ± 0.0132	27
	50 Hz, 16, 10^{-3}	782,434	0.7201 ± 0.0084	20
	50 Hz, 16, 10^{-4}	782,434	0.6942 ± 0.0074	24
Time Series (output features, learning rate)	16, 10^{-3}	12,866	0.7548 ± 0.0130	9
	16, 10^{-4}	12,866	0.7327 ± 0.0307	17
	64, 10^{-3}	74,978	**0.7826** ± 0.0178	7
	64, 10^{-4}	74,978	0.7597 ± 0.0143	8
Synchronized Time Series (resampling frequency, output features, learning rate)	20 Hz, 16, 10^{-3}	12,850	0.7343 ± 0.0194	16
	20 Hz, 16, 10^{-4}	12,850	0.6941 ± 0.0248	25
	20 Hz, 64, 10^{-3}	74,914	0.7509 ± 0.0123	12
	20 Hz, 64, 10^{-4}	74,914	0.7466 ± 0.0166	14
	50 Hz, 16, 10^{-3}	12,850	0.7247 ± 0.0110	18
	50 Hz, 16, 10^{-4}	12,850	0.6860 ± 0.0113	26
	50 Hz, 64, 10^{-3}	74,914	0.7486 ± 0.0211	13
	50 Hz, 64, 10^{-4}	74,914	**0.7511** ± 0.0201	10
Handcrafted Features (output features, learning rate)	14, 10^{-3}	10,000	**0.6699** ± 0.0170	29
	14, 10^{-4}	10,000	0.6672 ± 0.0185	32
	64, 10^{-3}	70,050	0.6683 ± 0.0151	31
	64, 10^{-4}	70,050	0.6693 ± 0.0181	30

Table A.4: Mean and standard deviation (separated by ±) of accuracy for task recognition on Buffalo keyboard data. Results are sorted by the encoding method and the last column indicates the ranking of the configuration relative to the others. There are three classes and the portion of the majority class is 60.11 %. Batch size was reduced from 128 to 64 for configurations annotated with (*) since they allocated too much GPU memory (> 10 GB) otherwise.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature