Karlsruhe Institute of Technology

Carnegie Mellon University

On-line Recognition of Handwritten Mathematical Symbols

Bachelor's Thesis of

Martin Thoma

At the Department of Informatics
Institute for Anthropomatics and Robotics (IAR)
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany

School of Computer Science
Interactive Systems Lab (ISL)
Carnegie Mellon University (CMU)
Pittsburgh, United States

Reviewer:	Prof. Dr. Alexander Waibel
Second reviewer:	Dr. Sebastian Stücker
Advisor:	Kevin Kilgour
Second advisor:	Prof. Dr. Florian Metze

Duration: June 2014 – November 2014

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text.

Karlsruhe, 07.11.2014

. . . .

(**Martin Thoma**)

Acknowledgement

Daniel Kirsch published the data collected with Detexify under the ODbL.[1] This dataset made it possible to evaluate many algorithms. Thank you Daniel!

My advisors Kevin Kilgour and Sebastian Stücker told me to make use of GPUs which boosted neural network training a lot. Thank you!

The StackExchange community helped me with very specific questions I had when I got problems with my implementation (StackOverflow) or regarding LaTeX (tex.stackexchange). Especially David Carlisle, Enrico Gregorio and percusse helped me to understand how LaTeX works, to get some of the diagrams to compile and helped me with a formulation in the introduction. Thank you!

Lara Martin and Anna Blomley helped me to notably improve the language in the first two chapters and the last chapter. It is now much easier to read and sounds much better. Additionally, I've learned a little bit about punctuation. Thank you, Lara and Anna!

The Baden-Württemberg Stiftung and interACT gave me the great possibility to write this bachelor's thesis at Carnegie Mellon University. Thank you!

interACT

Studienstiftung
des deutschen Volkes

Baden-Württemberg
STIPENDIUM

Ein Programm der

Baden-
Württemberg
Stiftung

WIR STIFTEN ZUKUNFT

[1] https://github.com/kirel/detexify-data

This work can be cited the following way:

```
@Misc{Thoma:2014,
    Title     = {On-line Recognition of Handwritten Mathematical Symbols},

    Author    = {Martin Thoma},
    Month     = {11},
    Year      = {2014},
    School    = Karlsruhe Institute of Technology,
    Address   = "Karlsruhe, Germany",
    Type      = "{B.S. Thesis}"

    Keywords  = {handwriting recognition; on-line; machine learning;
                 artificial neural networks; mathematics; classification;
                 supervised learning; MLP; multilayer perceptrons; hwrt;
                 write-math},
    Timestamp = {2014.06.07},
    Url       = {http://martin-thoma.com/write-math}
}
```

A DVD with a digital version of this bachelor's thesis and the source code as well as the used data is part of this work.

Abstract

Finding the name of an unknown symbol is often hard, but writing the symbol is easy. This bachelor's thesis presents multiple systems that use the pen trajectory to classify handwritten symbols. Five preprocessing steps, one data multiplication algorithm, five features and five variants for multilayer Perceptron training were evaluated using 166 898 recordings which were collected with two crowdsourcing projects. The evaluation results of these 21 experiments were used to create an optimized recognizer which has a TOP1 error of less than 17.5 % and a TOP3 error of 4.0 %. This is improvement of 18.5 % for the TOP1 error and 29.7 % for the TOP3 error.

Contents

1. Introduction

Euclid's *Elements* is one of the oldest mathematical texts that is still available. It was written in 300 BC by the ancient Greek mathematician Euclid. At that time, it was not possible to replicate information fast. Since a person had to copy the book by hand, its creation was relatively simple regarding the technology being used, but it was difficult to spread information.

The invention of the printing press changed this, and in 1482, Euclid's Elements was first set in type. By using a plate, ink, and a press, one could easily make hundreds of copies. However, the creation of the plate was difficult. It was made with a combination of movable metal types that could be reused for other texts and wooden templates for formulas and drawings. In summary, it can be said that the printing press made it easy to replicate information once the plate was created, but creating it was hard.

The creation of the original text became easier with the invention and evolution of computers, and the possibilities for replication became cheaper and more effective. With computers, one can easily restructure chapters with just a few keystrokes. Words, and even whole paragraphs, can simply be inserted or deleted wherever the author wants. Modern, low-priced printers can easily print 20 pages per minute, and the Internet can be used to spread information on a scale that was unimaginable before. TeX, a language that allows typesetting of almost arbitrary content was initially released by Donald Knuth in 1978. It got extended by LaTeX and is still available for free. It offers to people the possibility, not only to create texts themselves, but also typeset them to a high standard without knowledge of typesetting algorithms.

Despite all of this progress, there is still a lot of potential to improve the process of writing. LaTeX code is written using a keyboard in a combination of Latin script and special characters like `{`, `}`, and `\` to form commands such as `\begin{equation}` or `\alpha`. One tedious task that all people learning LaTeX have to do to find the code for the symbol they want to write. This can be done by looking in symbol tables. However, as touch devices become ubiquitous, systems can be created to let users write a symbol, record it, and output the LaTeX command of the recognized symbol. This task of finding a proper textual representation of a given handwritten symbol is called handwriting recognition (HWR). If the recognition software only uses the pixel image of the recording, it is called off-line HWR. On-line HWR can use information from how the symbols were written, which includes the pen trajectory.

On-line HWR can use techniques of off-line HWR, but studies have shown that on-line information notably improves recognition rates and simplifies algorithms [BN72, GAC^{+}91].

This thesis is about on-line HWR. The type of machine learning task is a classification task, meaning that the set of symbols which should be recognized is provided.

1.1. Symbols, Glyphs and LaTeX Codes

A *symbol* is an atomic semantic entity which has exactly one visual appearance when it is handwritten. Examples of symbols are: $\alpha, \propto, \cdot, x, \int, \sigma, \dots$ [1]

While a symbol is a single semantic entity with a given visual appearance, a glyph is a single typesetting entity. Symbols, glyphs and LaTeX commands do not relate:

- Two different symbols might have the same glyph. For example, the symbols `\sum` and `\Sigma` both render to Σ, but they have different semantics and hence they are different symbols. Other symbols that have the same or similar glyphs can be found in table B.2.
- Two different glyphs might correspond to the same semantic entity. An example is `\varphi` (φ) and `\phi` (ϕ): Both represent the small Greek letter "phi", but they exist in two different variants. Hence `\varphi` and `\phi` are two different symbols.
- Examples for different LaTeX commands that represent the same symbol are `\alpha` (α) and `\upalpha` ($\upalpha$): Both have the same semantics and are hand-drawn the same way. This is the case for all `\up` variants of Greek letters.

It is also worth noting that LaTeX commands are neither always glyphs nor always single symbols. The LaTeX command `\ll` renders to $\ll$, which are two symbols. More examples of LaTeX commands that generate two or more symbols can be found in table 3.1 on page 9.

1.2. MathML and LaTeX

The task of symbol recognition is independent of the recognized symbol's output language as long as the output language is powerful enough.

Both MathML and LaTeX can be used to express a lot of formulas. The difference between them is how they were meant to be used. LaTeX was developed as an input language, that is, people should be able to easily write what they want to express. MathML, on the other hand, is an XML format and hence is easier for programs to parse.

Converters can transform one format into the other. A simple LaTeX-to-MathML converter can be found at `http://www.mathtowebonline.com` and a MathML-to-LaTeX converter is given by XSLT at `http://code.google.com/p/web-xslt/source/browse/trunk/pmml2tex/`.

LaTeX is used in this bachelor's thesis because it is easier to read. One can expect readers to understand the LaTeX command `\varphi` but not the Unicode code point `\u03C6`. As one aim of this bachelor's thesis is to provide a symbol recognition system that can be used to find the code for a hand-drawn symbol, the semantically meaningful output `\varphi` is of higher use for the user than `\u03C6`.

Also, LaTeX can be used to express any mathematical formula due to its powerful extension system. Every common symbol can be expected to be in at least one package, as LaTeX has been around for over 30 years now and — as shown by submissions to `arxiv.org` — is still used a lot.

A notable downside of LaTeX is that parsing it is hard. Even simple tasks — like checking if a symbol appears in the rendered output of a given text — is not trivial with LaTeX.

[1] The first symbol is an `\alpha`, the second one is a `\propto`.

1.3. Steps in Handwriting Recognition

One possible way in which handwriting recognizers can work is by performing the following steps in order to recognize characters, symbols, or words. Not every recognizer uses all of these steps.

1. **Preprocessing**: Recorded data is never perfect. Devices have errors and people make mistakes while using devices. To tackle these problems there are preprocessing algorithms to clean the data. The preprocessing algorithms can also remove unnecessary variations of the data that do not help classify but hide what is important. Having slightly different sizes of the same symbol is an example of such a variation. Nine preprocessing algorithms that clean or normalize recordings are explained in section 3.2.
2. **Data multiplication**: Learning algorithms need lots of data to learn internal parameters. If there is not enough data available, domain knowledge can be considered to create new artificial data from the original data. Ideas for data multiplication in the domain of on-line handwriting recognition can be found in section 3.3.
3. **Segmentation**: The task of formula recognition can eventually be reduced to the task of symbol recognition combined with symbol placement. Before symbol recognition can be done, the formula has to be segmented. As this bachelor's thesis is only about single-symbol recognition, this step was not evaluated.
4. **Feature computation**: A feature is high-level information derived from the raw data after preprocessing. Some systems like Detexify, which was presented in [Kir10], simply take the result of the preprocessing step, but many compute new features. This might have the advantage that less training data is needed since the developer can use knowledge about handwriting to compute highly discriminative features. Various features are explained in section 3.4.
5. **Feature enhancement**: Applying principal component analysis (PCA), linear discriminant analysis (LDA), or feature standardization might change the features in ways that improve the performance of learning algorithms. Section 4.1 describes feature standardization.

After these steps, we are faced with a classification learning task which consists of two parts:

1. **Learning** parameters for a given classifier. This process is also called *training*.
2. **Classifying** new recordings, sometimes called *evaluation*. This should not be confused with the evaluation of the classification performance which is done for multiple topologies, preprocessing queues, and features in Chapter 6.

Two fundamentally different systems for classification of time series data were evaluated. One uses greedy time warping, which has a very easy, fast learning algorithm which only stores some of the seen training examples. The other one is based on neural networks, taking longer to train, but is much faster in recognition and also leads to better recognition results.

1.4. Limitations of Single-Symbol Recognition

The recognition capabilities of single-symbol classifiers have some limitations that multi-symbol classifiers do not have. There are symbols such as the multiplication dot "·" versus the point ".", or zero "0" versus the capital and the small Latin letter "O" and "o" which can be distinguished by context and the availability of a baseline, but are extremely hard if not impossible, to distinguish without context. For example, a preceding “1” can indicate if the current symbol is a “0” or an “O”. More examples of symbols that look identical without context are given in table B.2.

As the design of write-math.com was set up without a ruled writing space, it is impossible to distinguish symbols that only differ in size or their relative position to a baseline. A baseline, and some context in terms of size and position, could have been established with a user interface like the one shown in figure 1.1. However, this was not done for two reasons: On the one hand, most data which was used is from the Detexify project which has neither this kind of single-symbol context nor a baseline. On the other hand, users with mobile devices should not be forced to write at an uncomfortably small size.

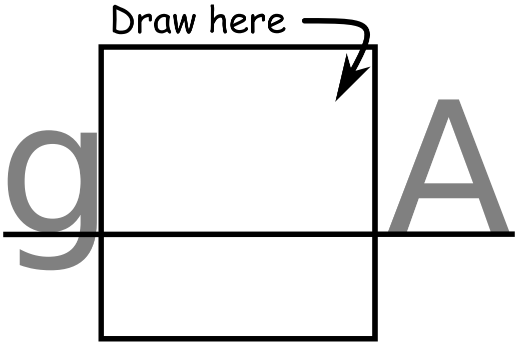


Figure 1.1.: *An example of how the user interface for single-symbol recognition could be designed. The advantage of this design over a simple empty box is that it gives the user some context as to how big he should write and where the baseline is. This information could later be used by a recognizer to distinguish “·” from “.” or “o” from “O”.*

2. Related Work

On-line handwriting recognition has been a field of study since T. L. Dimond developed a device for reading handwritten characters in 1958 [Dim58]. In the past 56 years, technology changed a lot. Computers went down in size from hundreds of square meters to less than half a square meter. The energy consumption and the weight were also notably reduced. At the same time, computing power grew exponentially. Computers became available for everybody. Multi-core processors started to spread in the early 2000s, and more data than ever were produced and stored in the world wide web. Graphics processing unit (GPU)-accelerated computing became usable with the Compute Unified Device Architecture (CUDA) platform, which was initially released in 2007, boosting the practical capabilities of neural networks. Combined with the enormous amount of data that is available through the Internet and services like Amazon Mechanical Turk, it becomes possible to design systems which learn from large amounts of data.

Meanwhile, there was also progress in the field of on-line handwritten mathematical formulas:

In 1966, G. F. Groner proposed a real-time recognizer that made use of a tablet that had a time-resolution of 4 ms and an accuracy of about 0.1 mm [Gro66]. The system recognized symbols by comparing sequences of the directions of strokes with labeled training data and applying manually-designed tests to features. His system was capable of recognizing 53 symbols, but only 52 symbols were used in the evaluation. The evaluation showed that the average recognition rate was at 87 %, but the users were instructed on how to use the system before the evaluation was done. This implies that the way users entered the symbols was perhaps not always the way they would normally write.

In the following years, a lot of work was done in cursive handwriting recognition. Jaeger, Manke, and Waibel described in [JMW00, JMRW01] a system that uses a multi-state time delay neural network (TDNN), which achieved recognition rates of over 90 % with context bitmaps for individual lowercase letters (a–z), individual uppercase letters (A–Z), or digits (0–9). Context bitmaps show a 3×3 bitmap of the proximity of a point.

One of the early works done in on-line handwriting recognition for mathematical formulas is [BH84]. Their system used a combination of sequence vectors and a feature that was the ratio of the distance from the starting point to the end point and the symbol height. Only 35 different symbols were evaluated. With those settings, a recognition rate of 93 % was achieved.

In 1998, A. Kosmala and G. Rigoll designed a system for on-line mathematical handwriting recognition which was trained to recognize 100 different symbols [KR98]. This included the 52 lowercase and uppercase letters, 23 mathematical symbols, 11 lower-case Greek letters, and 6 parentheses. The system was designed to recognize complete formulas, although the symbols of the formula had to be drawn in a predefined order. It applied hidden Markov models (HMMs) for symbol segmentation. The data was resampled, but no other preprocessing was described. A sampled bitmap was used as a feature, as well as on-line features like the writing direction. One-hundred common mathematical and physical formulas were used as a training set, and 30 additional formulas as a test set. They claimed to get recognition rates of 96.3 %. However, this seems to be very high since in the Competition on Recognition of Online Handwritten Mathematical Expressions (CROHME) from 2013, the best team achieved recognition rates of 60.36 % and the second best team achieved recognition rates of 23.40 % as documented in [MVGZ+13]. The first two competitions, [MVGK+12] and [MVGK+11], also did not receive any submissions that had recognition rates over 65 %.

Daniel Kirsch used over 1000 symbols in his diploma thesis [Kir10]. He evaluated a very simple recognition system called *Detexify* which was — and still is, at the time of this work — accessible through the web.[1] Many people can use the recognition system as nothing else than a browser and internet access is required, while providing a huge number of symbols that can get classified. However, in his evaluation, Kirsch used only a randomly-chosen subset of 100 symbols. He claimed to get a TOP1 error of 26.12 % and a TOP3 error of less than 10 % with standard dynamic time warping (DTW) and other variants of DTW.

The aim of this work is to build a recognition system that is as accessible as Detexify, but is faster, can recognize more symbols, and has a higher recognition rate. The presented system is able to classify 369 symbols. Furthermore, this work contributes to the first publicly-available dataset for on-line handwriting recognition, with more than 166 898 recordings. This will help to make experiments and different classifiers comparable. The symbol classifier A (see section 4.2) can be tested on `write-math.com`, and the comparably better classifier B (see section 6.7) is currently not publicly available but is planned to be released in the near future.

[1] `http://detexify.kirelabs.org/`

3. Domain Specific Classification Steps

Taking a close look at the collected data might give relevant insights into problems one has to deal with and eventually imagine preprocessing steps that can reduce those problems. It could also lead to ideas for features that are invariant to variations that occur in the dataset.

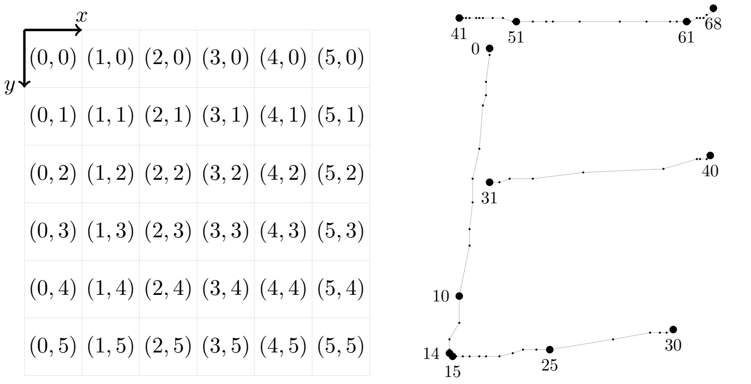
This chapter explains classification steps that are specific for on-line HWR, whereas the next chapter explains the rather domain independent task of classification of time series data.

3.1. Data

The data that was used for all experiments was collected with write-math.com and detexify.kirelabs.org (see [Kir10]). write-math.com is a website designed by me for this bachelor's thesis whereas Detexify was created by Daniel Kirsch. Both websites use HTML and JavaScript to gather data and both websites store the same data, but in a slightly different data format.

write-math.com makes use of HTML5 canvas elements. Those elements can be used in combination with JavaScript to track fingers or a mouse cursor touching the canvas, moving and lifting. Every point is specified by two integer coordinates (x, y). The origin $(0, 0)$ is at the upper left corner of the rectangular canvas element and x values get bigger to the right and y values get bigger to the bottom. Figure 3.1a shows such an HTML5 canvas plane. JavaScript asynchronously triggers events that contain the information where on the canvas the cursor or finger currently is. Those points are called *control points* in the following. For the mouse, the information if the mouse button is currently pressed down is also available. So when the position is recorded, the stroke-wise segmentation is automatically given for both, mouse and fingers. A list of such user generated control points together with the information which points belong to the same stroke and the information when the point was recorded is called a *recording*. An example of a recording is figure 3.1b.

91.93 % of the 166 898 recordings that were used in the evaluation were collected by Detexify. The recordings are stored in JSON format as a list of strokes. Each stroke consists of tuples $(x(t), y(t), t)$ where x and y are canvas coordinates and t is a timestamp given in milliseconds since 1970. An example of a recording in JSON format is in section E.



$(0,0)$	$(1,0)$	$(2,0)$	$(3,0)$	$(4,0)$	$(5,0)$
$(0,1)$	$(1,1)$	$(2,1)$	$(3,1)$	$(4,1)$	$(5,1)$
$(0,2)$	$(1,2)$	$(2,2)$	$(3,2)$	$(4,2)$	$(5,2)$
$(0,3)$	$(1,3)$	$(2,3)$	$(3,3)$	$(4,3)$	$(5,3)$
$(0,4)$	$(1,4)$	$(2,4)$	$(3,4)$	$(4,4)$	$(5,4)$
$(0,5)$	$(1,5)$	$(2,5)$	$(3,5)$	$(4,5)$	$(5,5)$

(a) HTML5 canvas plane (b) Recorded sequence of points

Figure 3.1.: *On the left side is an HTML5 canvas plane. Each coordinate $(x, y) \in \mathbb{N}_0^2$ is one pixel. Every coordinate has to be non-negative and an integer. On the right side is a visualization of a recording after preprocessing steps that reduced the number of points. The small and the large points are the control points. The large points are points with an annotation which indicates the order in which the points were recorded. Point* 0 *is the first point that was recorded, point* 68 *is the last one. Points of one stroke were connected with straight lines. Figure 3.1b has 4 strokes in total.*

The time resolution between points as well as the resolution of the recording depends on the device that was used. However, most recordings have a time resolution of about 20 ms and are within a bounding box of a 250 px × 250 px square. Figure 3.2 shows how the time between control points is spread amongst the analyzed data. It shows that one can expect a time resolution of 50 ms and should eventually treat control points of one stroke that take longer as errors.

166 898 recordings were collected for the 369 classes which were tested.[1]

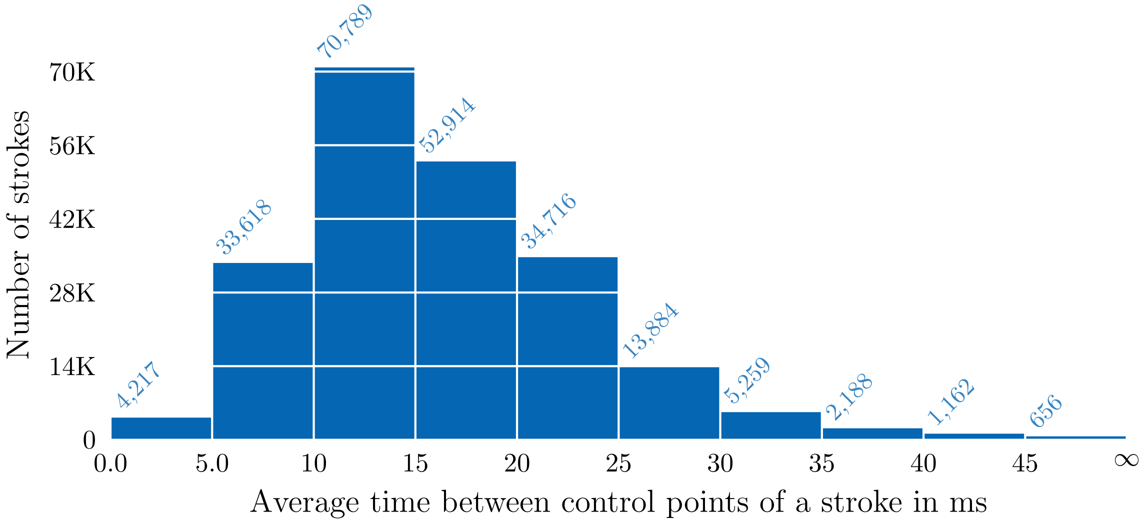


Figure 3.2.: *More than* 98 % *of all time values between two control points of the same stroke are less than* 35 ms. *More than* 73 % *are captured faster than in* 20 ms.

[1] Links to those recordings and more are available at martin-thoma.com/write-math.

3.1.1. Choice of Symbols

The choice of symbols which the classifier was trained to recognize was directly influenced by the number of obtained recordings per symbol. None of the 431 symbols with less than 50 recordings were evaluated, although some of them are used in mathematical formulas.

The following symbols or groups of symbols were then removed from the remaining set of 680 symbols:

- Symbols that don't fit in the context of this work:
  - Text mode-only symbols: `\MVAt` (**@**), @, `\textsurd` (√), . . .
  - Image-like symbols: `\Bat` (🦇), `\Mundus` (🌍)
- LaTeX commands that are not symbols as defined before:
  - "\big. . ." variants: `\bigoplus` ($\bigoplus$), `\bigstar` ($\bigstar$), `\bigcup` ($\bigcup$) . . .
  - "\Up" and "\up" variants of Greek letters: `\Upsigma` (Σ), `\uppi` (π), `\uplambda` (λ), . . .
  - "\thick" variants: `\thicksim` ($\thicksim$), `\thickapprox` ($\thickapprox$)
  - `\dotsb` ($\cdots$), but `\dots` was evaluated
  - `\cdotp` ($\cdot$) because it is the same as `\cdot` ($\cdot$), except that it is used for punctuation whereas `\cdotp` is used for the binary math operator.
  - `\ocircle` ($\circledcirc$) because it is the same as the included symbol `\circledcirc` ($\circledcirc$).
  - Multiple-symbol LaTeX commands like `\ll` ($\ll$) as shown in table 3.1. In a multiple-symbol classifier, these symbol sequences could be detected and replaced in a post-classification step.

Search		Replace	
LaTeX	Rendered	LaTeX	Rendered
`\int\int`	$\int\int$	`\iint`	$\iint$
`\int\int\int`	$\int\int\int$	`\iiint`	$\iiint$
`\int\int\int\int`	$\int\int\int\int$	`\iiiint`	$\iiiint$
`<<`	$<<$	`\ll`	$\ll$
`<<<`	$<<<$	`\lll`	$\lll$
`>>`	$>>$	`\gg`	$\gg$
`>>>`	$>>>$	`\ggg`	$\ggg$
`\int\cdots\int`	$\int\cdots\int$	`\dotsint`	$\int\cdots\int$

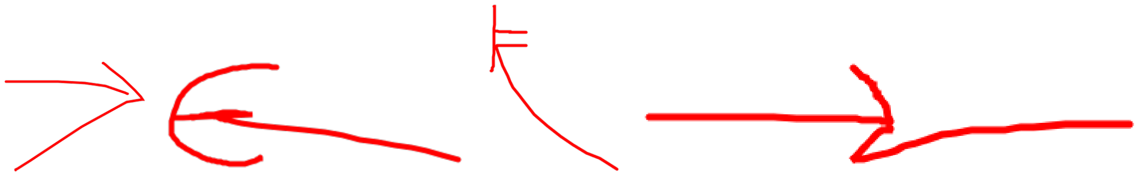
Table 3.1.: *The single-symbol LaTeX commands shown above have a pendant which renders to multiple symbols. A multiple-symbol classifier could search for those recognized patterns and replace them by a single LaTeX command in order to get a better typesetted version of the text. That reduces the number of classes such a multiple-symbol classifier has to be able to recognize.*

All symbols that were used to evaluate the algorithms are listed in tables B.10 to B.18. This includes:

$a - z$	Small letters	$\alpha - \omega$	Small Greek letter	$\rightarrow, \leftarrow, \Rightarrow, \Leftarrow, \Leftrightarrow, \dots$
$A - Z$	Capital letters	$A - \Omega$	Capital Greek letters	$=, \sim, \equiv, \approx, \dots$
$0 - 9$	Digits		$+, -, \cdot, \surd, \cup, \cap, \dots$	$\oplus, \star, \dots$

(a) ID 288612 ($\in$) (b) ID 291939 ($\forall$) (c) ID 282212 ($\models$) (d) ID 262502 (Π)

Figure 3.3.: *Examples for missing strokes (problem D2). The classification was added by the user who created the recording. It is not possible to tell if the captured single point was the last or the first point of a stroke.*



(a) ID 258177 (b) ID 270115 (c) ID 286813 (d) ID 249024

Figure 3.4.: *Examples for too long strokes (problem D3) that users probably did not want to make that long.*

3.1.2. Problems

As the data was collected via crowdsourcing it has errors. Human classification errors are only a problem for model training; a model trained with these might make the same error as humans made before. Four different types of human classification errors can be distinguished:

H1 *Confusion*: Recordings were classified wrong, but the correct class looks similar to the chosen class, e.g. ϵ, ε and $\in$.

H2 *Creativity*: Drawings that should not have been entered in the first place were arbitrarily classified by the user. Some examples are shown in figure D.3.

H3 *Cherry-Picking*: Drawings of complete formulas were entered and classified as a class of a single symbol of that formula.

H4 *Manipulation*: Obviously wrong classified symbols, e.g. ϵ that gets classified as α.

Additionally to those human classification errors, there are errors that are caused by the device or the human who uses it while drawing. Those errors should be considered in preprocessing:

D1 *Wild points*: Points that appear randomly anywhere on the drawing plane.

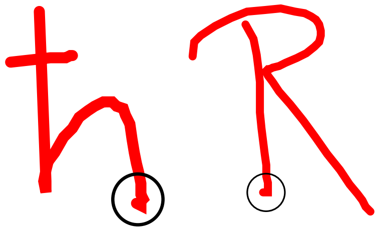
D2 *Missing strokes*: The user drew a stroke, but only the first point or the last point was captured. This might happen more often when the user tries to draw small strokes with his fingers. Examples are shown in figure 3.3.

This problem could be confused with problem D1.

D3 *Too long strokes*: The user made a stroke much longer than he wanted to. Examples are shown in figure 3.4.

D4 *Hooks*: At the beginning or end of a stroke the user makes a hook, which he did not want to make. Examples are shown in figure 3.5.

D5 *Interrupted strokes*: Although the user drew one stroke, the stroke is interrupted and thus recorded as multiple strokes. See figure 3.9a on page 16 for an example.



(a) ID 8350 (b) ID 11387

Figure 3.5.: *Examples for hooks at the end or the beginning of a stroke that should not be there (problem D4).*

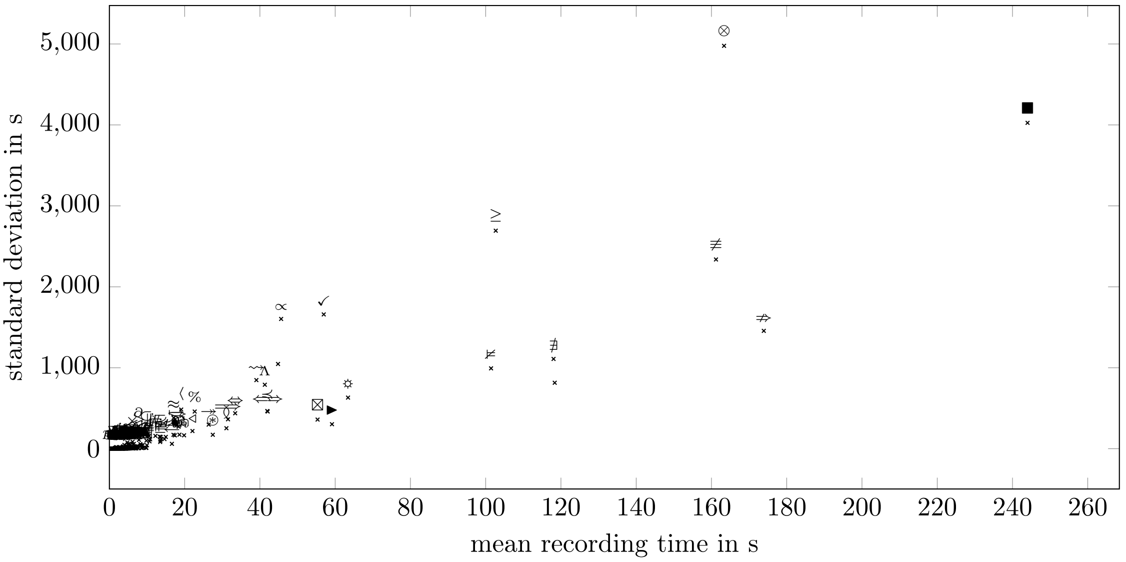


Figure 3.6.: *Mean and standard deviation of the recording time of symbols in seconds. Almost every symbol has a recording time of less than* 20 s *and a standard deviation of less than* 500 s*. The high standard deviation indicates that there are some recordings with extremely wrong timestamps. Some recordings have a huge gap between two subsequent control points. The points seemed to be still in order, but the data looked as if the system clock was changed while the symbol was drawn.*

D6 *Multiple drawn strokes*: Some people draw strokes twice. This introduces new variants how symbols can be drawn. See figure 3.9d on page 16 for an example.

D7 *Wrong timestamps*: Some of the data seems to have the wrong time. It seems highly unlikely that users took over 10 minutes to draw a single symbol, yet alone over a day. A plot for which the mean recording time and the standard deviation of every symbol is shown in figure 3.6 and the four most extreme values in table 3.2.

Other problematic user actions are:

O1 *Filling areas*: Filling areas produces a lot of points. But the order and the number of those points is arbitrary in contrast to many — eventually all — other strokes. See figure 3.9b on page 16 as an example for a recording with a filled area.

O2 *Strengthened strokes*: Sometimes users want to “strengthen” strokes. As with problem O1, the number of those strengthening points might vary a lot even for a single user. See figure 3.9c on page 16 as an example for a recording with a strengthened stroke.

Symbol	Mean	std deviation	Symbol	Mean	std deviation
`\boxdot`	2864 ms	$31.76 \cdot 10^6$	`\nsubseteq`	1994 ms	$26.86 \cdot 10^6$
`\subsetneq`	1199 ms	$18.27 \cdot 10^6$	`\psi`	324 ms	$7.04 \cdot 10^6$

Table 3.2.: *Mean and standard deviation of the recording time in milliseconds of symbols that are not shown in figure 3.6.*

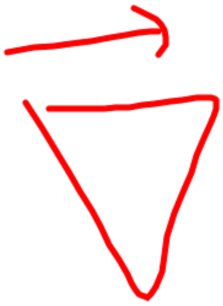


Figure 3.7.: ∇ *written by the user "Marienkaefer". It is an example where the user expects the system to recognize something different than the closest LaTeX pendant* $\vec{\nabla}$.

All recordings that suffered from problems problems H1 to H4 were excluded from the evaluated dataset. For problem H2, the recording was additionally marked as an image or as a member of the "trash" class. The trash class was neither used for training nor for evaluation, but it could be used in future to detect if a user wants to delete a recording he just drew.

Recordings that were multiple symbols (problem H3) were additionally annotated with the number of symbols for future complete formula recognition.

Problems D1 to D5 are covered by automatic methods which are explained in section 3.2. Problem D7 was ignored.

One reason why problem H1 (symbol confusion) and problem H4 (manipulation) are very difficult to note and to resolve is that users might write something different when they use handwriting compared to what they use in printed text. One example is the following: In physics, it seems to be common to write $\vec{\nabla}$ in handwritten text, but use ∇ in LaTeX. In that case, the classifier should recognize figure 3.7 as ∇, although the appearance is closer to $\vec{\nabla}$.

3.1.3. Data Cleansing

The data was collected by crowdsourcing. There were no restrictions and everybody could enter data anonymously. In the case of Detexify, where over 91.93 % of the data comes from, this happened over 4 years.

This means a lot of the data is classified wrong.

In the case of the test set, all recordings were checked manually. But there is too much data to manually check all recordings. So different techniques were used to automatically find suspicious recordings.

The greedy time warping classifier, which is explained in section 4.2, was used to find recordings with a high distance within all recordings a single symbol. The distance of every recording to every other recording of the same symbol was measured. This means when a symbol had n recordings, there were $n \cdot (n-1)$ time warpings done. Then the recordings were ordered descending by distance. They were reviewed until at least 10 recordings in a row were classified correct.

The global features were used to find outliers. For every global feature in section 3.4, the mean and the standard deviation of every symbol was calculated. The symbols with highest standard deviation were examined. For those symbols, the recordings were ordered descending and reviewed until at least 10 recordings in a row were classified correct.

Neural network classifiers were trained and the errors they made were examined for misclassified recordings.

All results in chapter 6 were obtained after the data cleansing steps.

3.2. Preprocessing

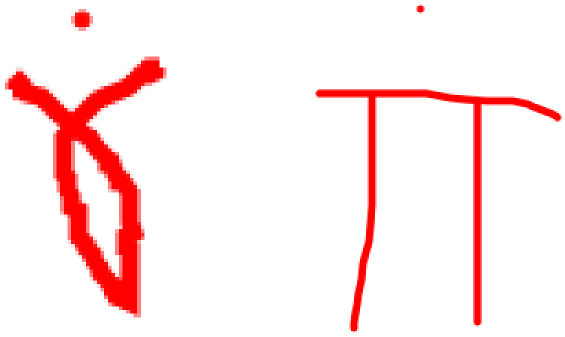
Preprocessing in symbol recognition is done to improve the quality and expressive power of the data. It should make follow-up tasks like segmentation and feature extraction easier, more effective or faster. It does so by resolving errors in the input data, reducing duplicate information and removing irrelevant information.

3.2.1. Normalization: Scaling, Shifting and Resampling

Scaling — which is also called *size normalization* — is done by many handwriting recognition systems, but the way in which size normalization is done varies.
Single-symbol recognizers such as the one presented in [Kir10] scale the data points to fit into a unit square while keeping their aspect ratio. To do so, the bounding box of the symbol is taken and everything is scaled according to this bounding box. Afterwards, the points are shifted to the $[0, 1] \times [0, 1]$ unit square. It was shown in [HZK09, Kir10] that this kind of preprocessing notably boosts classification accuracy.
[GAC⁺91] shifts the symbol to $[-1, 1] \times [-1, 1]$. That might be better for the training of neural networks as it might lead to a mean feature value of 0 (see section 4.1 for more information).
An algorithm that does scaling and shifting to $[-0.5, 0.5] \times [-0.5, 0.5]$ while keeping the aspect ratio is given in pseudocode on page 68. Three implementation variants of the scale and shift algorithm are explained and evaluated on page 38.

Everything that makes the recording artificially bigger makes scaling less effective. That includes wild points (problem D1) and hooks (problem D4). Algorithms that can deal with those problems are described in section 3.2.2.

Another method to normalize data is *resampling*. This is called *stroke length normalization* in [TSW90]. [GAC⁺91] resampled characters and digits to 81 points each, where different strokes were connected by “pen-up” segments. They resampled to get points regularly spaced in arc length, not in time. [JMRW01] also resampled the points to be equidistant in space, but they used a distance of $\frac{\text{corpus height}}{13}$. They found an improvement of 5 % with this preprocessing step. [SGH94] also resampled data to get points regularly spaced in arc length, but they encoded speed as an extra feature. A simple resampling algorithm that interpolates strokes linearly and spaces points equidistant in time for a fixed number of points. Algorithm 2 on page 67 shows this simple resampling algorithm in pseudocode.



(a) Raw data ID 149550 (b) Raw data ID 138361

Figure 3.8.: *Examples of recordings with a dot over the symbol. It is not possible to tell if that is a wild point or a decoration which was intended by the user.*

3.2.2. Noise Reduction

The following list of noise reduction techniques was created by [TSW90] and is still up-to-date.

- **Dot reduction** reduces dots to single points. Sometimes multiple points get recorded although the user wanted to make only a single point, e.g. for one of the following symbols: $\cdot$, $.$, $\dots$, $\vdots$, $\ddots$, i, $\therefore$, $\because$. This can be detected by calculating the maximum distance d two points in a stroke have. If d is smaller than a threshold, then it is a single point. In that case all points of the stroke get reduced to a single dot. This dot could be the center of mass of all points in the stroke. The algorithm can be found in pseudocode on page 68.

- **Dehooking** is the removal of hooks (see problem D4) which the author did not want to write. Hooks appear sometimes at the beginning or the end of strokes. Examples can be seen in figure 3.5. An algorithm for dehooking is described in [HZK09].

- **Filtering** is the process of removing points by some criteria. Those criteria include:

  - Duplicate points as applied in [HZK09, GP93],
  - Enforcing a minimal distance between consecutive points [TSW90].
  - Maximum velocity / acceleration [Tap87]
  - Enforcing a minimal change in direction [TSW90].

  Occasionally occuring control points that were generated by device errors are one reason to apply a filtering preprocessing step. Those points are also called *wild points* (problem D1). Filtering wild points might be difficult for humans when the points could also be decorations as shown in figure 3.8.

  One way to detect wild points is by measuring the speed from the last point to the wild point. If that speed is too high, it can be assumed that it is a wild point.

- **Smoothing** can be done in at least two ways. An approach that was used quite often is applying a weighted average [Gro66, Tap87, Ara83]. Algorithm 6 describes in pseudocode how weighted average smoothing can be implemented.

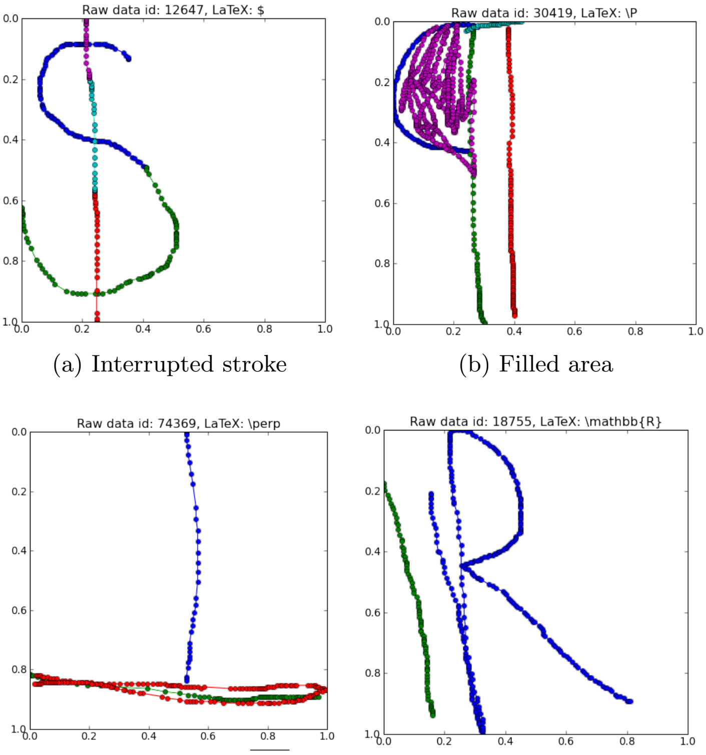
  It takes three weighting parameters $\theta_1, \theta_2, \theta_3 \in [0, 1]$ and recalculates the point coordinates of every point p_i except the first point p_1 and the last point p_n like this:

$$p_i' \leftarrow \theta_1 \cdot p_{i-1} + \theta_2 \cdot p_i + \theta_3 \cdot p_{i+1}$$

Another way to do smoothing would be to reduce the number of points with the Douglas-Peucker algorithm to the most relevant ones and then interpolate those points. The Douglas-Peucker stroke simplification algorithm is usually used in cartography to simplify the shape of roads. The Douglas-Peucker algorithm works recursively to find a subset of control points of a stroke that is simpler and still similar to the original shape. The algorithm adds the first and the last point p_1 and p_n of a stroke to the simplified set of points S. Then it searches the control point p_i in between that has maximum distance from the line $p_1 p_n$. If this distance is above a threshold ε, the point p_i is added to S. Then the algorithm gets applied to $p_1 p_i$ and $p_i p_n$ recursively. Pseudocode of this algorithm is on page 69. It is described as “Algorithm 1” in [VW90] with a different notation.

- **Connecting strokes** should be done if problem D5 (see page 10) occurs. This can be detected by measuring the distance between the end of one stroke and the beginning of the next stroke. If this distance is below a threshold, then the strokes are connected.
  [GP93] describes that such maliciously disconnected components can get detected by observing angular continuity and the shortness of distance between two strokes. The distance between two consecutive strokes (s_i, s_{i+1}) is calculated by measuring the euclidean distance from the last point of s_i to the first point of s_{i+1}. As this error seems just to split strokes, but not miss any control point, it might result in control points of subsequent strokes being very close. So one could also use only the distance and a distance threshold to determine if two strokes should be connected.

- **Deskewing** corrects character slant. Although this technique was applied by some authors [BS89, GP93, HBT94], it seems not to be applicable to the domain of mathematical handwriting, because on the one hand symbols might occur in variations with slant, like $\rightarrow$ and $\nearrow$. On the other hand it is questionable if slant is as consistent with symbols as it is with cursive handwriting.



(a) Interrupted stroke

(b) Filled area

(c) Strengthened stroke

(d) Multiple stroke drawing

Figure 3.9.: *Every image shows a recording after scaling and shifting by visualizing the stored points and connecting them with a straight line. Points and the lines between them are colored with the same color if they belong to the same stroke and otherwise with a different color.*

3.2.3. Order of Preprocessing Steps

There are multiple dependencies regarding the order in which the mentioned preprocessing steps should be executed:

- Duplicate point removal is dot reduction with any minimum distance > 0.
- Dot reduction should be done before wild point filtering is done, because multiple points might get reduced to a single dot. Hence wild point detection might improve, because the reduced dot is isolated a little bit more.
- The scaling step depends on the size of the bounding box. As wild point removal and smoothing could change that size, those two algorithms should be applied before smoothing gets applied.
- Everything that changes the number of points should be done before resampling. That includes (wild) point filtering and smoothing.

Those dependencies and the preprocessing parameters are visualized in figure 3.10.

Table B.1 lists all presented preprocessing algorithms with the range of their parameters.

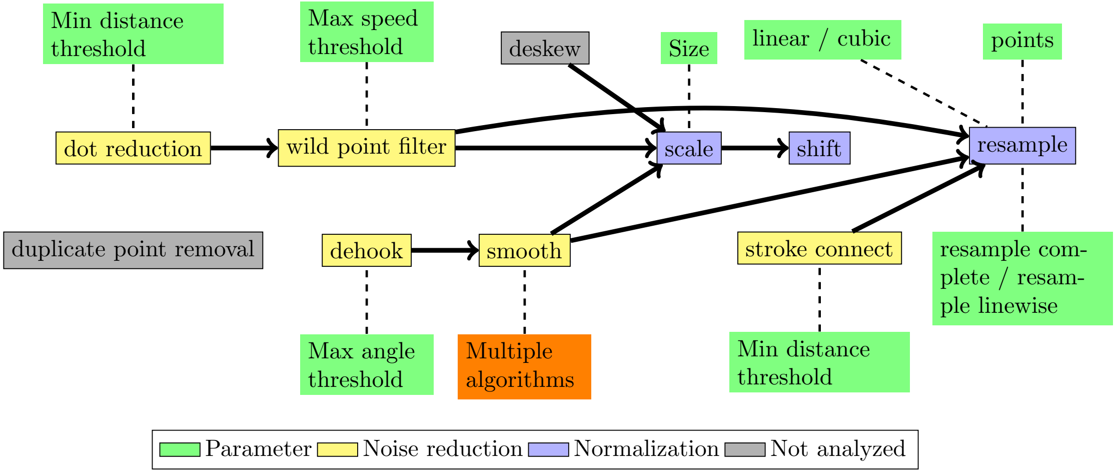


Figure 3.10.: *It makes sense to order the application of preprocessing algorithms — if they are applied at all — as shown in the diagram. The noise reduction algorithms are yellow, normalization algorithms are blue, green are parameter. Smoothing has too many variants and parameters to show them all in this diagram. Deskewing was not analyzed and is very likely not applicable in the domain of on-line handwritten recognition of single mathematical symbols. Duplicate points removal is only a special variant of dot reduction.*

3.3. Data Multiplication

Obtaining a lot of original data can be difficult. Although projects like Amazon Mechanical Turk might help, one could eventually still see the need of more data. One way to get more data and to make the classifier invariant to some transformations is by giving “virtual examples” that incorporate those invariances [SBV96]. That means domain knowledge is used to artificially generate more data from original data.

For on-line handwriting recognition, invariant transformations could be

- Rotation by a maximum degree in the range of $(-22.5°, 22.5°)$ as symbols like $\rightarrow$ and $\nearrow$ are already transformations of 45°. The rotation center could be the center of mass (arithmetic mean of coordinates)
- Small random movements of single points independently from other points. However, this has to be used very carefully because of symbols like $\rightarrow$, $\rightsquigarrow$ and $\leadsto$ where those movements could easily lead to recordings that cannot be distinguished.
- Scaling with or without respect to the aspect ratio. This might also have side effects like π and Π or $\rightarrow$, $\rightarrow$ and $\longrightarrow$.

Other variations like scaling with respect to the aspect ratio or shifting do only make sense when the preprocessing algorithm that removes those invariances is not used. The use of data multiplication algorithms can break invariances created by preprocessing steps. An example is that after applying a scaling algorithm, one expects all recordings to have the same bounding box size. However, after a recording was rotated that is no longer the case.

3.4. Features

A number of different features have been suggested for on-line handwriting recognition. They can be grouped into local features and global features. Local features apply to a given point on the drawing plane and sometimes even only to point on the drawn curve whereas global features apply to a complete stroke or even the complete recording.

3.4.1. Local Features

The following local features were used for on-line handwriting recognition. However, most features were used as part of a bigger system without evaluating the effect of the single feature.

- **Coordinates** of the current point are used by [GAC⁺91].
- **Speed** has been used by [SGH94], but [KR98, KRLP99] suggest that speed is a bad feature, because they think that speed is “highly inconsistent”.
- **Binary pen pressure** has been used by [KR98, KRLP99, SGH94, MFW94, GAC⁺91].
- **Direction** has been used by [MFW95, HK06]. The **direction** at the point i can be described by the vector $(\cos\theta(i), \sin\theta(i))$ as described in [GAC⁺91]:

$$\cos\theta(i) = \frac{\Delta x^{(i)}}{\Delta s^{(i)}} \tag{3.1}$$

$$\sin\theta(i) = \frac{\Delta y^{(i)}}{\Delta s^{(i)}} \tag{3.2}$$

where

$$\Delta x(i) = x^{(i+1)} - x^{(i-1)} \tag{3.3}$$

$$\Delta y(i) = y^{(i+1)} - x^{(i-1)} \tag{3.4}$$

$$\Delta s(i) = \sqrt{(\Delta x(i))^2 + (\Delta y(i))^2} \tag{3.5}$$

- **Curvature** has been used by [Gro66, MFW95, SGH94, GAC+91]. It is calculated in [GAC+91] by the angle of two neighboring lines like this:

$$\varphi(i) = \theta(i+1) - \theta(i-1) \tag{3.6}$$

$$\cos\varphi(i) = \cos\theta(i-1) \cdot \cos\theta(i+1) \tag{3.7}$$

$$+ \sin\theta(i-1) \cdot \sin\theta(i+1) \tag{3.8}$$

$$\cos\varphi(i) = \cos\theta(i-1) \cdot \cos\theta(i+1) \tag{3.9}$$

$$- \sin\theta(i-1) \cdot \sin\theta(i+1) \tag{3.10}$$

- **Bitmap-environment** has been used by [MFW94]. This feature is a 3 × 3 pixel environment around the current point. It allows the recognizer to determine points that cross or touch strokes. Adding this feature reduced the error by 50 % compared to using only coordinates, the direction, curvature and speed.
- **Hat-Feature** has been used by [SGH94, JMW00].

3.4.2. Global Features

- **Re-curvature** is defined in [HK06, HZK09] as the ratio between the height of a stroke and the distance between its start and end point. It is not clear if this distance was meant to be the euclidean distance or the distance on the stroke. Both variants were tried, but the distance on the stroke gives much better evaluation results. So it was chosen to use the feature

$$\text{re-curvature}(\mathit{stroke}) = \frac{\text{height}(\mathit{stroke})}{\text{length}(\mathit{stroke})}$$

- **Center point** for every single stroke was used in [HK06]. A center point of a stroke is the arithmetic mean of the coordinates.
- **Stroke length** was used in [HK06]. It can be calculated by using the summed line segment length after a linear interpolation step.
- **Number of strokes** was used in [HZK09].
- **Sequence features**
  - *Pen-tip sequence*: [Kir10] used the raw pen-tip sequence combined with DTW variants to recognize mathematical symbols. Other authors like [KWL95] used pen-tip sequences, too, but made use of HMMs or artificial neural networks (ANNs) to recognize symbols.
  - *Zone sequences* are used by [Bro64, iHY80]. The idea is to recognize symbols by dividing the box in which the character is written into zones. By examining the position of the pen-tip a sequence of zones can be generated for a written symbol.
  - *Direction sequences* were used in [IMP76, Pow73].

- **Aspect ratio** of the bounding box of the recording.

There are other global features used for off-line handwriting recognition which will not be examined. Examples are Pseudo-Zernike moments and Shadow Code features which were used in [KC98].

4. Domain Independent Classification Steps

The previous chapter shows the used data as well as preprocessing steps and features that can be found in on-line HWR. This chapter introduces some general methods that can be applied in any classification task of time series data. At this point we have pairs of feature vectors $x \in \mathbb{R}^n$ and class labels y. The set of those pairs (x, y) is split into three distinct subsets: A training set, a validation set and a test set. The training set can be used by a learning algorithm to adjust internal parameters. However, the training algorithm could be able to adjust too much and create a recognizer that works well on the training set but much worse on new examples. Hence the validation set is used to detect when the algorithm suffers from overfitting. The test set on the other hand only gets used when the training is finished and the system can be evaluated.

Although a lot of learning algorithms like Gaussian mixture models (GMMs), support vector machines (SVMs), k-Nearest Neighbors and even more can be applied for classification tasks, only two are explained and evaluated: Greedy time warping (GTW) and multilayer perceptrons (MLPs). The GTW classifier is easy to implement and works reasonably well with only a few training examples, but it is slow in evaluation. MLPs on the other hand are harder to implement, take longer to train, but evaluate new data faster and with higher recognition rates if enough data is available as showed in chapter 6.

4.1. Feature Enhancement

Feature enhancement algorithms can be used to make the already calculated features more useful for training algorithms. The effect of those algorithms depends on both, the data and the used learning algorithm.

An important subset of the feature enhancement algorithms are those that reduce the dimensionality. PCA and LDA are such algorithms.

One simple feature enhancement is *feature standardization* sometimes also called *feature normalization*. For some learning algorithms it is useful if the different features have a mean of 0 and either a similar range or a similar variance. Feature standardization gives this property.

Feature standardization is done by calculating the mean $\overline{x}$ of all feature vectors $x \in T$ in the training set T. Then, before the training gets applied and before every evaluation, the mean gets subtracted from every feature vector x_i:

$$x' \leftarrow x_i - \overline{x}$$

This is called *mean normalization.* In order to standardize features one has to divide x' by the range of values $\max(T) - \min(T)$ of the training set.

If the feature is only divided by either the range or the variance it is called *feature scaling.*

4.2. Greedy Time Warping

A web system for on-line handwritten symbol recognition was implemented and is described in [Kir10]. It uses an GTW algorithm which is similar to DTW.

The idea of the GTW algorithm is to calculate how far the points between two recordings A and B have to be moved to match each other. The algorithm calculates a distance $d(A, B)$ of two recordings. This is done with help of the squared euclidean distance δ.

In the following a_i denotes the ith point of the recording A and b_i the ith point of the recording B. α_i denotes the number of the point in the recording A that was moved in step i and β_i denotes the number of the point in recording B that was moved in step i.

The distance $\delta(a_{\alpha_0=0}, b_{\beta_0=0})$ between the first points of A and B is calculated. Then the minimum of $\delta(a_{\alpha_{i+1}=\alpha_i+1}, b_{\beta_{i+1}=\beta_i})$, $\delta(a_{\alpha_{i+1}=\alpha_i+1}, b_{\beta_{i+1}=\beta_i+1})$ and $\delta(a_{\alpha_{i+1}=\alpha_i}, b_{\beta_{i+1}=\beta_i+1})$ is added to the already calculated distance. If $\alpha_i + 1$ does not exist because α_i is already the number of points in A then only the last distance is taken. Similar, if $\beta_i + 1$ does not exist because β_i is already the number of points in B then only the first distance is taken.

Pseudocode is on page 70.

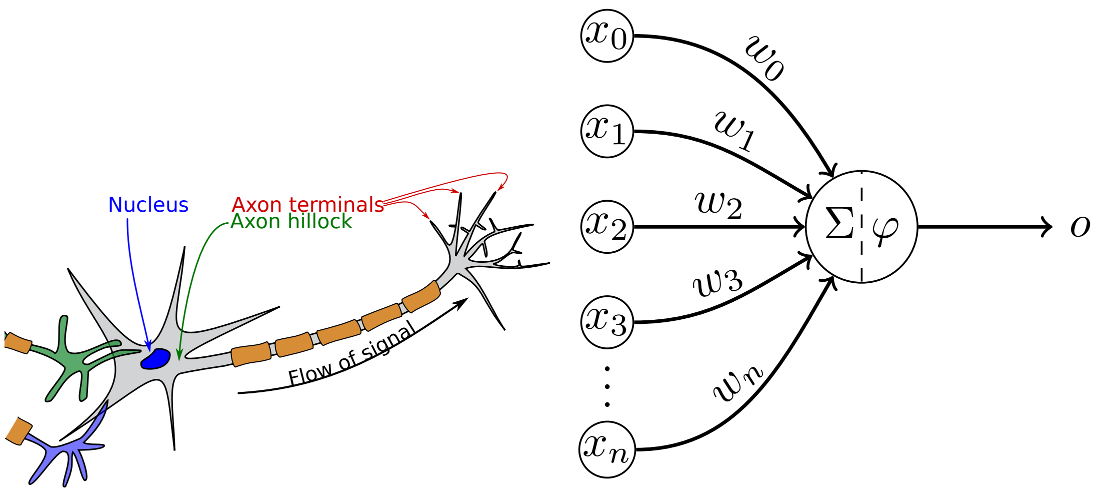
4.3. The Perceptron Algorithm

The idea of developing an algorithm that has similar capabilities as the brain probably began in 1943 when Warren McCulloch and Walter Pitts described the binary threshold unit in [MP43]. This work was later continued by Frank Rosenblatt who invented the perceptron algorithm in 1958 [Ros58]. The perceptron is a function

$$\begin{aligned} p_{w,\varphi} :&\mathbb{R}^n \rightarrow \mathbb{R} & p_{w,\varphi}(x) :&= \varphi(w^T \cdot x) \\ \varphi :&\mathbb{R} \ \ \rightarrow \mathbb{R} & \varphi(x) :&= \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

This function, or rather the visualization of it as shown in figure 4.1b, is also called an *artificial neuron.* Artificial neurons are inspired by biological neurons such as the one illustrated in figure 4.1a. In biological neurons, signals are sent within the cell by charged particles, so called *ions.* But before a biological neuron sends a signal, a threshold charge has to be reached at the axon hillock. This threshold charge is called *action potential.* The action potential can be reached by multiple factors, but the one which is most interesting are charges send by other neurons. The closer other axon terminals are to the axon hillock, the more their signal contributes to reaching the action potential. If the stimulated neuron has reached the action potential, it sends a signal.

Artificial neurons are similar as they receive input signals and give an output signal. Those input signals get weighted and summed up. Then an activation function φ is applied to the weighted sum. However, there are important differences, too. Artificial neurons use



(a) Biological Neuron

(b) Artificial Neuron

Figure 4.1.: *Both neurons receive weighted input signals, apply a function to that sum and send an output signal.*

different activation functions. In most applications, artificial neurons use a differentiable function which sends a continuous signal whereas a biological neuron encodes the information by the frequency it sends a signal. Biological neurons send signals asynchronously, but PCs work synchronously. More details can be found in [LBK$^+$08, p. 1001–1026] and [CRU$^+$08, p. 1047–1061].

An application of the perceptron is a binary classifier where the parameters $w \in \mathbb{R}^n$ have to be learned. In the context of supervised learning there are already m training examples of input vectors $x_i \in \mathbb{R}^n, i \in \{ 1, \dots, m \}$ together with the desired output $y \in \{ 0, 1 \}$ given. The output is called the *class* and $x_i^{(j)}$ is the jth *feature* of the ith training example.

When such a training set $T = \{ (x_i, y_i) \mid i \in \{ 1, \dots, m \} \}$ is given, we want to find a choice for w that is best for that set according to an differentiable error function $E : \mathbb{R}^n \to \mathbb{R}_0^+$. The error function E can be modified to represent not only the error on the training set, but also additional targets. Therefore it is also called loss function, objective function or cost function. [Mit97, p.89–92] describes in detail how the perceptron learns its weight parameters w.

We want to find the minimum of that function E. One way to find the minimum of a function is by gradient descent. That means one starts at a random point w, calculates the gradient at this point and “goes” in the direction of the gradient, that means the weights are adjusted. This is commonly expressed as

$$w \leftarrow w + \Delta w$$

and hence this learning method is called *delta rule*. In this case w and Δw are vectors where the single vector components are

$$\Delta w^{(j)} = -\eta \frac{\partial E}{\partial w^{(j)}}$$

where $\eta \in \mathbb{R}_{>0}$ is called the *learning rate*. The training algorithm will overshoot the minimum if it is too big, but when it is too small, the training algorithm will make progress very slow.

A common way to visualize gradient descent is to imagine the error surface. It is a surface in the $\mathbb{R}^{n+1}$ where n dimensions are the possible choices of the parameter $w \in \mathbb{R}^n$ and the last dimension is the error $E(w) \in \mathbb{R}$. The form of that surface depends on the training examples and the error function. As the error function uses the output of the perceptron, it depends on the activation function. It follows that the activation function φ has to be differentiable. Hence the sign function is not a good choice. A common choice for φ is the *sigmoid function*:

$$\text{sigmoid}(x) := \frac{1}{1+e^{-x}}$$

The perceptron classifier is able to make use of an arbitrary number of features to distinguish two classes.

However, in the case of symbol classification there are more than two classes. One way to solve this is by applying the *one-vs.-rest* strategy. That means for every class there is one classifier that tests if the recording belongs its class. When a recording should get classified, the output of every single neuron gets calculated. Then the softmax function gets applied to the vector of outputs o of those neurons.

$$\text{softmax} : \mathbb{R}^n \rightarrow [0,1]^n \qquad \text{softmax}(o)^{(i)} := \frac{e^{o^{(i)}}}{\sum_{j=1}^{n} e^{o^{(j)}}}$$

The softmax function makes sure that every single value of the result is in $[0, 1]$ and that the sum of all values is exactly 1. Furthermore, the order of the values in that vector remains the same. One could say that the softmax function transforms a vector of scores to a vector of probabilities.

Another mayor drawback of a single layer perceptron is the fact that it can only classify data which is linearly separable in the feature space. The feature space is usually an $\mathbb{R}^n$, where n is the number of features. Every recording of the training data is a point in that space. While the obtained data might usually be in the $\mathbb{R}^2$ or $\mathbb{R}^3$, the features might give relationships between this information. By a clever choice of features one can make data that was not linear separable in the obtained space separable in the feature space. In fact, one can make every training set linearly separable by adding a new feature per training example that gives the distance to that training example. But that would be a lot of features and the resulting model would very likely suffer from overfitting. As the number of training examples might be very high, even dimensionality reduction algorithms like PCA and LDA could be difficult to apply.

For this reason it is desirable that the neural network is able to learn features by itself. This can be achieved by using multiple layers, where every layer computes a new set of features.

4.4. Multilayer Perceptron

A MLP is organized in layers of artificial neurons. Every artificial neuron is a function $p_{w,\varphi}$ with different weight vectors w per artificial neuron. Every layer has exactly one activation function φ, but the activation functions of different layers may be different. Every layer is fully connected with its predecessor and its successor.

The number of layers is in principle not limited and the number of neurons is not limited either. However, the number of parameters between a layer i with n_i neurons and a layer j with n_j neurons is $n_i \cdot n_j$. That means for subsequent layers with many neurons the number of parameters that have to be learned gets very big.

4.4.1. Notation

A notation that is almost identical to the one in [Mit97] was chosen:

- $\ell \in \mathbb{N}$ is the number of layers of the MLP.
- $n_j \in \mathbb{N}$ is the number of neurons in layer $1 \leq j \leq \ell$.
- $(x_i, y_i) \in \mathbb{R}^{n_1} \times \mathbb{R}^{n_\ell}$ is a single training example of the training set T.
- $v^{(i)}$ is the ith element of a vector v.
- $x_{j,i} \in \mathbb{R}$ is the ith input to the j neuron.
- $w_{j,i} \in \mathbb{R}$ is the weight from neuron i to neuron j.
- $\text{net}_j := \sum_{i \in \text{input neurons}} w_{j,i} x_{j,i}$ is the activation of neuron j, that means the value that the activation function is applied to.
- $o_j(x) := \varphi_j(\text{net}_j(x)) = \varphi_j(\sum_{i \in \text{input neurons}} w_{j,i} x_{j,i})$ is the output of neuron j after the MLP got x as input feature. If j was not in the last layer, there is at least one i such that $o_j = x_{i,j}$ (note the order).
- *outputs* is the set of all neurons in the last layer (the output layer).
- $D(j)$ is the *Downstream*, that means the set of all neurons that have neuron j as a direct input. That means the downstream of j includes all neurons of the layer that is nearer to the output layer directly after the layer in which the neuron j is.

Figure 4.2 visualizes the notation.

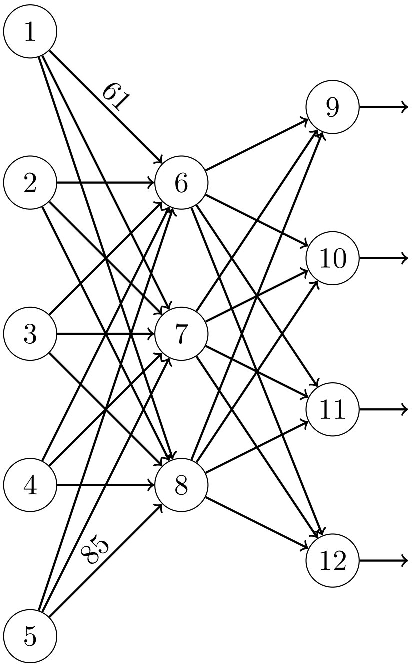


Figure 4.2.: *Visualization of the used notation. Every neuron has an index (1–12). Only two weights were labeled:* $w_{6,1} = 61$ *and* $w_{8,5} = 85$. *The downstream of neuron 6 is* $D(6) = \{ 9, 10, 11, 12 \} = D(7) = D(8)$. *If this was a 3 layer perceptron where the neurons 9, 10, 11 and 12 are the output layer, then outputs* $= \{ 9, 10, 11, 12 \}$.

4.4.2. Activation Functions

The activation function of artificial neurons have to be differentiable and their derivative has to be non-zero so that the gradient descent learning algorithm can be applied. At least one layer should also be non-linear, because linear combinations of linear functions are again linear functions. So if all activation functions of a MLP were linear, the complete MLP would only represent a linear function. This means the neural network could be reduced to a MLP without any hidden layer.

The last layer in classification tasks is often the softmax function. For all other layers it is often the sigmoid function and sometimes also the hyperbolic tangent tanh. The advantage of tanh over the softmax function is that it converges faster when the absolute value of the argument is big.

Figure 4.3 shows the activation functions sigmoid, tanh and the sign function.

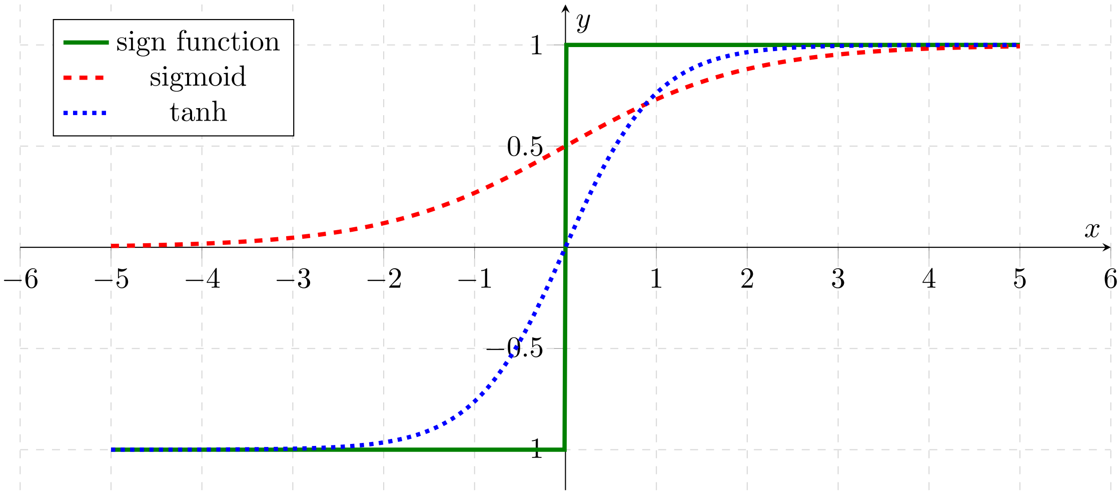


Figure 4.3.: *A plot of the sign function, the sigmoid function and the hyperbolic tangent. All three functions can be used as activation functions in artificial neurons.*

4.4.3. Evaluation

The evaluation of a neural network is very similar to the evaluation of a single perceptron. For every perceptron of the first layer, the weights are multiplied with the input. Those values are added and then the activation function gets applied. This is repeated until the output of the first layer completely calculated. Then exactly the same process is repeated with every following layer.

However, this evaluation can also be expressed with matrix multiplications. The input vector $x_I \in \mathbb{R}^{n_i - 1}$ gets extended by one value to the vector $x \in \mathbb{R}^{n_i}$. This value is 1 and represents the bias. Then the vector x is multiplied by weight matrix $W \in \mathbb{R}^{n_i \times n_j}$ resulting in a vector a which is also called *activation*:

$$x^T \cdot W = a^T \in \mathbb{R}^{n_j}$$

After that, all activation functions get applied point-wise to the activation vector a to get the output vector o with the output of every neuron of that layer.

The advantage of this matrix-wise expression is that some programs can automatically parallelize this multiplication and that GPUs can compute those matrix multiplications directly.

4.4.4. Supervised Training with Gradient Descent

The gradient descent algorithm is a supervised algorithm for training MLPs. Just like the perceptron algorithm in section 4.3 it needs an error function which can be minimized. Cross entropy (CE) is a possible choice for MLPs with a softmax output layer:

$$\begin{aligned} E_{x_i} :& \mathbb{R}^{n_1 \times n_2} \times \mathbb{R}^{n_2 \times n_3} \times \dots \times \mathbb{R}^{n_{\ell-1} \times n_\ell} \to \mathbb{R}_{\geq 0} \\ E_{x_i}(W) :=& -\sum_{k=1}^{n_\ell} \left(y_i^{(k)} \log(o(x_i)^{(k)}) + (1 - y_i^{(k)}) \log(1 - o(x_i)^{(k)}) \right) \\ E_B :& \mathbb{R}^{n_1 \times n_2} \times \mathbb{R}^{n_2 \times n_3} \times \dots \times \mathbb{R}^{n_{\ell-1} \times n_\ell} \to \mathbb{R}_{\geq 0} \\ E_B(W) =& \sum_{x_i \in B} E_{x_i}(W) \end{aligned}$$

where E_{x_i} is the error for a single training example and E_B with $\emptyset \neq B \subseteq T$ is called a *mini-batch*. Different choices of B lead to different training modes as explained in section 4.4.5.

There are other error functions like the classification figure of merit (CFM) or mean squared error (MSE) [HW89]. However, in the following describes only the training with the CE function.

The error function E_B is to be minimized. The gradient descent algorithm with batch gradient descent converges to a local minimum if the learning rate is decreased while applying gradient descent multiple times.

As the error is the sum of non-negative values, we get a lower error by minimizing the error for every single training example if the learning rate η is low enough. However, it should be noted that those minimizations are not independent. This means the global error could increase with single stochastic gradient descent and single mini-batch gradient descent steps, although the learning rate is low.

The training algorithm is

Algorithm 1 Stochastic Gradient Descent

```
function TRAIN($T$, $W$)
    for epoch ← 1; epoch ≤ 1000; epoch ← epoch + 1 do
        for all $(x, t_x) \in T$ do
            for all weights $w_{j,i}$ do
                $w_{j,i} \leftarrow w_{j,i} - \eta \frac{\partial E_{\{x\}}}{\partial w_{j,i}}(W)$
```

where the number of epochs could be adjusted or changed to another stopping criterion like a threshold for the change in validation error or the value of the cost function.

Computing the partial derivatives $\frac{\partial E_B}{\partial w_{j,i}}$ is not a trivial task, but it is explained in detail in [Mit97].

Finally, the weight update rule can be formulated as

$$w_{j,i} \leftarrow w_{j,i} + \Delta w_{j,i} \tag{4.1}$$

$$\Leftrightarrow w_{j,i} \leftarrow w_{j,i} + \eta \delta_j x_{j,i} \tag{4.2}$$

where δ_j is a term that depends on the layer and is recursively defined. For CE as an error function, an output layer that makes use of the softmax activation function and sigmoid activation functions in all hidden layers it is

$$\delta_j = \begin{cases} \frac{1}{|B|} \sum_{x_i \in B} \left(y_i^{(j)} - o_j(x_i) \right) & \text{if } j \in \mathit{outputs} \\ \frac{1}{|B|} \sum_{x_i \in B} \left(o_j(x_i)(1 - o_j(x_i)) \sum_{k \in D(j)} \delta_k w_{k,j} \right) & \text{otherwise} \end{cases}$$

The δ_j get calculated layer-wise, starting from the output layer. This is the reason why this learning algorithm is also called the *backpropagation algorithm*, although it is only a special case of gradient descent. The signal gets propagated through the network, the output is generated and then the error is propagated back.

4.4.5. Batch, Mini-Batch and Stochastic Gradient Descent

Neural Networks can be trained in three different training modes. The *stochastic gradient descent* takes one training example and adjusts the weights. Another training mode is *mini-batch gradient descent* where a chunk of a fixed size b, the size of the mini-batch, is used to calculate the gradient and to adjust the weights. A third training mode is *batch gradient descent* where all training examples are used to calculate the adjustment of weights. A common choice for the mini-batch size is $b = 256$. However, for $b = 1$ it is stochastic gradient descent and for $b = |T|$ it is batch gradient descent. The advantage of stochastic gradient descent is that the weights are updated faster, compared to batch gradient descent. The advantage of batch gradient descent is that weight updates are more meaningful. Mini-batch gradient descent can be faster than stochastic gradient descent, because the weights are updated less often.

4.4.6. Momentum

One problem of simple gradient descent is the choice of the learning rate. Depending on how much the error changes between different epochs, one might choose a higher learning rate or lower it. A learning parameter called *momentum* tries to implement such an automatic adjustment of the error.

If one imagines the error surface in the parameter space, one can imagine the current weight as a ball. The ball begins to roll down the error surface. If it does not change the direction much and keeps rolling down, it speeds up. If the direction changes or if the weight increases, the momentum decreases. It also keeps the ball going in the direction that worked before. So in case of an error surface that has a plateau, the momentum helps to get away from that plateau.

The momentum $\alpha \in [0, 1]$ changes the weight update to

$$\Delta w_{j,i(\text{epoch}_i)} = \eta \delta_i x_{j,i} + \alpha \Delta w_{j,i}(\text{epoch}_{i-1})$$

as described in [Mit97, B$^+$06].

4.4.7. Newbob Training

Newbob training is an adaptive training that is described in [new00]. It starts with a learning rate η and trains until the error on the validation set decreases by less than $\theta_1 = 0.5\,\%$. When that happens, the learning rate is multiplied with a decay parameter. `newbob-decay` $= 0.5$ is chosen in [new00]. The training is stopped when the error drops by less than $\theta_2 = 0.5\,\%$ after the threshold θ_1 was hit in the training step before. Those two thresholds can be adjusted, of course.

4.4.8. Denoising Auto-encoder

An *auto-encoder* is a neural network that is trained to restore its input. This means the number of input neurons is equal to the number of output neurons. The weights are an *encoding* of the input that allows restoring the input. As the neural network finds the encoding by itself, it is called auto-encoder. If the hidden layer is smaller than the input layer, it can be used for dimensionality reduction [Hin89]. If only one hidden layer with linear activation functions is used, then the hidden layer contains the principal components after training [DHS01].

Denoising auto-encoders are a variant introduced in [VLBM08] that is more robust to partial corruption of the input features. It is trained to get robust by adding noise to the input features.

There are multiple ways how noise can be added. Gaussian noise and randomly masking elements with zero are two possibilities. [Deea] describes how such a denoising auto-encoder with masking noise can be implemented. The `corruption` is the probability of a feature being masked.

4.4.9. Pretraining

When a neural network gets more layers, the number of weights can decrease even if the total number of neurons increases. For example, a MLP with a 500 : 500 : 500 topology has

$$500^2 + 500^2 = 500\,000$$

weights, but a MLP with a 500 : 100 : 500 : 500 topology has

$$500 \cdot 100 + 100 \cdot 500 + 500^2 = 350\,000$$

weights.

However, the more weights a MLP gets, the more random initializations are done for this MLP. This might lead to high variations in classification performance for the same training queue, but different weight initializations. One possible way to deal with this problem is to apply pretraining. This means that the layers are trained before the layers get stacked to form the resulting model. This means at first, the first hidden layer gets trained. Then the first two layers get trained, etc.

Pretraining can be done supervised, semi-supervised or unsupervised. A supervised training algorithms needs labels for all training examples, an unsupervised does not use any labels and a semi-supervised needs labels for some examples, but not for all.

Denoising auto-encoders are an example for unsupervised pretraining. Supervised layer-wise pretraining (SLP) is to train first a MLP with one hidden layer, then discard the output layer, add the second hidden layer and a new output layer and train again.

4.4.10. Regularization

Regularization is a group of methods that help to prevent overfitting, that means the problem that a model performs much worse on the test set than on the training set. The idea of regularization in MLP is that sparse weights or weights with a low absolute value tend not to cause overfitting and are therefore preferred. This can be encoded in the training algorithm by modifying the cost function such that higher weights correspond with a higher cost when compared to lower weights that have a similar error on the training set.

Two common regularizations are L_1 and L_2 regularization. L_1 regularization adds the absolute value of the weights to the error function and L_2 regularization adds the squared parameters to the error function [Ng04].

5. Implementation

When this bachelor's thesis was written, there was no publicly available data set for on-line handwritten mathematical symbols. In order to get the necessary data to conduct experiments, the website `write-math.com` was created in preparation for this bachelor's thesis as a free-time project. The code for the website is available at `https://github.com/MartinThoma/write-math`. While data was gathered, Daniel Kirsch was contacted and asked for the data recorded by `detexify.kirelabs.org`. After some months, he published the data. A link to the data as well as a description of the data format is available at martin-thoma.com/write-math.

The following sections describe four different projects that were important for this bachelor's thesis:

- `write-math`: The website that was created to collect recordings
- `hwrt`: The toolkit to view recordings and make experiments
- `hwr-experiments`: The files that define the experiments
- Neural Network Training: An internal project for creation, evaluation and training of neural networks.

5.1. write-math.com

The website `http://write-math.com` was created to get data. It is a combination of PHP, MySQL, JavaScript, CSS and HTML. It makes use of the front-end framework Bootstrap and the template engine Twig. The source is available at `https://github.com/MartinThoma/write-math`.

The website allows the users to classify recordings (see figure G.4a on page 92). As soon as the user has drawn the symbol, he clicks on submit and gets redirected to a classification page. He sees the recording, get a link to a page where he can try out preprocessing methods and see the symbols that classifiers suggested. Every user has the possibility to add his own classifier that others are also able to use. Every time a new recordings gets submitted, the website contacts every known classifier by sending a JSON string via POST-request. The website expects every classifier to respond by serving a JSON string that contains a list of at most 10 dictionaries which map symbol identifiers (integers) to probabilities. This could look like

```
[{"31":0.88842893496419},
 { "1":0.10999419040225},
 {"36":0.001499575497246},
 {"40":7.7299136313199e-5}]
```

The list must be ordered descending by probability. Figure 5.1 visualizes this workflow.

Currently, only System A is online.

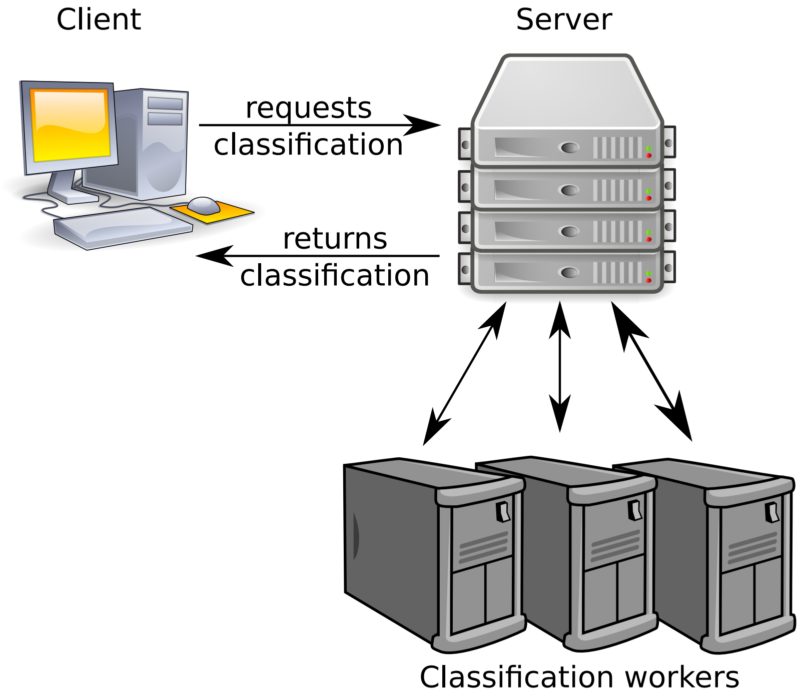


Figure 5.1.: *The workflow of a single classification is the following: (1) The user writes a symbol. This symbol gets recorded by the users browser via JavaScript and send to the server. (2) The server stores the recording and contacts all classification workers. (3) Each classification workers sends a list with up to 10 symbols and their probability back to the server. (4) The server stores those suggestions and shows the user all results. The image of a desktop computer on the top left is from `https://commons.wikimedia.org/wiki/File:Computer-aj_aj_ashton_01.svg` and was created by an unknown artist, the server image on the top right is from `https://commons.wikimedia.org/wiki/File:Server-multiple.svg` and was created by RRZEicons and the images that was used three times for classification workers is from `https://commons.wikimedia.org/wiki/File:Server_by_mimooh.svg` and was created by Mimooh.*

5.2. Handwriting Recognition Toolkit

A toolset was created for the analyzation, preprocessing and feature calculation of on-line handwritten data. This toolset was bundled in a Python module called `hwrt`. It is freely available over the Python Package Index (PyPI) and can be installed with `pip install hwrt`. It contains algorithms for preprocessing, feature selection and data multiplication as well as tools to download the latest data, view and analyze the data.

The following preprocessing algorithms were implemented. They all work on exactly one recording. They were described in section 3.2.

- `RemoveDuplicateTime`: If a recording has two points with the same timestamp, than the second point is discarded. This is useful for a couple of algorithms that don't expect two points at the same time.
- `RemoveDots`: Remove all strokes that have only a single point (a dot) from the recording, except if the whole recording consists of dots only.
- `ScaleAndShift`: Scale a recording so that it fits into a unit square. This keeps the aspect ratio. Then the recording is shifted. The default way is to shift it so that the recording is in $[0, 1] \times [0, 1]$. However, it can also be used to be centered within $[-1, 1] \times [-1, 1]$ around the origin $(0, 0)$ by setting `center=True` (for the smaller dimension) and `center_other=True` (for the bigger dimension).
- `SpaceEvenly`: Space the points evenly in time over the complete recording. The parameter `number` defines how many points should the recording should get in total. All strokes get connected by lines. All points on the strokes get a `pen_down=True` feature and all points between strokes get a `pen_down=False` feature.
- `SpaceEvenlyPerStroke`: Space the points evenly for every single stroke separately. The parameter `number` defines how many points are used per stroke and the parameter `kind` defines which kind of interpolation is used. Possible values include `cubic`, `quadratic`, `linear`, `nearest`. This part of the implementation relies on `scipy.interpolate.interp1d`.
- `DouglasPeucker`: Apply the Douglas-Peucker stroke simplification algorithm separately to each stroke of the recording. The algorithm has a threshold parameter `epsilon` that indicates how much the stroke is simplified. The smaller the parameter, the closer the resulting strokes are to the original.
- `StrokeConnect`: Detect if strokes were probably accidentally disconnected. If that is the case, connect them. This is detected by the threshold parameter `minimum_distance`. If the distance between the end point of a stroke and the first point of the next stroke is below the minimum distance, the strokes are connected.
- `DotReduction`: Reduce strokes where the maximum distance between points is below a `threshold` to a single dot.
- `WildPointFilter`: Find wild points and remove them. The threshold means speed in pixels / ms.
- `WeightedAverageSmoothing`: Smooth every stroke by a weighted average. This algorithm takes a list `theta` of 3 numbers that are the weights used for smoothing.

The following data multiplication algorithms were implemented. They were described in section 3.3.

- `Multiply`: Copy the data n times.
- `Rotate`: Adds rotational variants of the recording. It has three parameters: `min`, `max` and `num`. The algorithm adds `num` rotated variants of the recording to the dataset.

The following features were implemented. They were described in section 3.4.

- `ConstantPointCoordinates`: Take the first `points_per_stroke=20` points coordinates of the first `strokes=4` strokes as features. This leads to $2 \cdot \text{points\_per\_stroke} \cdot \text{strokes}$ features.

  If `points` is set to 0, the first `points_per_stroke` point coordinates and the `pen_down` feature is used. This leads to $3 \cdot \text{points\_per\_stroke}$ features.

  If there are not enough points or strokes, the feature gets filled with `fill_empty_with=0`.

- `FirstNPoints`: Similar to the `ConstantPointCoordinates` feature, this feature takes the first `n=81` point coordinates. It also has the `fill_empty_with=0` to make sure that the dimension of this feature is always the same.
- `StrokeCount`: The number of used strokes can be a powerful feature. Figure 6.4 (page 46) gives an impression how good this feature can separate some symbols.
- `Bitmap`: $n \times n$ grayscale bitmap or the recording, where `n` is a parameter. A human can recognize most recordings with $n = 32$ and still many with $n = 18$.
- `Ink`: Ink as a 1-dimensional feature. It gives a numeric value for the amount of ink this would eventually have consumed.
- `AspectRatio`: Aspect ratio $(\frac{\text{width}+0.01}{\text{height}+0.01})$ of a recording as a 1-dimensional feature.
- `Width`: Width of a recording as a 1-dimensional feature.
  Note that this is the current width. So if the recording was scaled, this will not be the original width.
- `Height`: Height of a recording as a 1-dimensional feature.
  Note that this is the current height. So if the recording was scaled, this will not be the original height.
- `Time`: The time in milliseconds it took to create the recording. This is a 1-dimensional feature.
- `CenterOfMass`: Center of mass of a recording as a 2-dimensional feature.
- `StrokeCenter`: Get the stroke center of mass coordinates for the first `stroke=4` strokes. The dimension of this feature is $2 \cdot \text{stroke}$.
- `StrokeIntersections`: Count the number of intersections which strokes in the recording have with each other in form of a symmetrical matrix for the first `stroke=4` strokes. The feature dimension is $\text{ROUND}(\frac{\text{strokes}^2}{2}) + \frac{\text{strokes}}{2}$, because the symmetrical part is discarded.
- `ReCurvature`: Re-curvature is a 1-dimensional, stroke-global feature for a recording. It is the ratio $\frac{\text{height}(\mathit{stroke})}{\text{length}(\mathit{stroke})}$.

5.3. Experiments

All experiments are saved as configuration files on `https://github.com/MartinThoma/hwr-experiments`. The handwriting recognition toolkit (HWRT) is able to use those configuration files and regenerate the models automatically. The structure of the configuration files is explained in section F.

5.4. Neural Network Implementation

The training and testing of neural networks with `hwrt` needs an executable `nntoolkit` that supports the following usages:

```
$ nntoolkit run --batch-size 1 -f%0.4f <test_file> < <model>
```

has to output the evaluation result in standard output as a list of floats separated by newlines `\n`. The evaluation result might either be the index of the neuron with highest activation or the list of probabilities of each class separated by spaces.

```
$ nntoolkit make mlp <topology>
```

has to print the model in standard output.

The `hwrt` toolset is independent of the way the training command is formatted as the training command gets inserted directly into the configuration file `info.yml` of the model.

In order to implement such a neural network executable one can use Theano, cuDNN (`https://developer.nvidia.com/cuDNN`) or Caffe (`http://caffe.berkeleyvision.org/`). `http://www.deeplearning.net/tutorial/` contains example code for multilayer perceptrons written with Theano (Python).

6. Evaluation

The following experiments and their results show how the previously described algorithms perform and how they influence the classification error on the test set. The training set has 134 804 recordings, the validation set has 15 161 recordings and the test set has 17 012 recordings. 369 symbols were tested. Those symbols are listed in tables B.10 to B.18.

All changes that are described in the following were done with 4 systems. All of those 4 systems use a simple preprocessing queue: Scaling with respect to the aspect ratio to fit into a unit square, shifting to $[-1, 1] \times [-1, 1]$ and linear resampling. The first 4 strokes of a recording were used for features, all other strokes were discarded. For each stroke, 20 points coordinates that were spread equidistant in time were taken as features. If a recording had less then 4 strokes, the feature got 0 as a value. Hence the trained neural networks gets $4 \text{ strokes } \cdot 20 \frac{\text{points}}{\text{stroke}} \cdot 2 \frac{\text{features}}{\text{point}} = 160$ input features which equals the number of input neurons.

System B_i has i hidden layers with 500 neurons per hidden layer. Mini-batch training with a batch size of 256, a learning rate of $\eta = 0.1$ and a momentum of $\alpha = 0.1$ was used. Every system B_i has a softmax layer at the end. Neither regularization nor pretraining were used. As different topologies might severely influence the classification results of MLPs, one baseline system was chosen for each of the 4 tested topologies.

Table 6.1 shows three types of errors for four different MLPs: *TOP1*, *TOP3* and *MER*. TOP n is the standard classification error which tests if the reference class was within the n hypotheses with highest probability. The MER error (short for *merged classes*) accepts the symbols in table B.2 as being equivalent. MER first gets the TOP3 hypotheses, extends this set M by all equivalent symbols and then checks if the reference class is within M.

System	Topology	Classification error		
		TOP1	TOP3	MER
B_1	160:500:369	23.34 %	6.80 %	6.64 %
B_2	160:500:500:369	21.51 %	5.75 %	5.67 %
B_3	160:500:500:500:369	21.93 %	5.74 %	5.64 %
B_4	160:500:500:500:500:369	23.88 %	6.12 %	6.04 %

Table 6.1.: *Evaluation of the baseline systems B_1–B_4 with three different classification error measures. All errors were measured on the test set.*

6.1. Influence of Random Weight Initialization

The neural networks in all experiments got initialized with a small random weight

$w \sim U(-4 \cdot \sqrt{\frac{6}{n_j + n_{j+1}}}, 4 \cdot \sqrt{\frac{6}{n_j + n_{j+1}}})$ where w is a weight between layer j and layer $(j+1)$

as suggested on [deeb]. The random initialization is done to break symmetry.

This might lead to different error rates for the same models just because the initialization was different.

In order to get an impression of the magnitude of the influence on the different topologies and error rates the baseline models were trained 5 times with random initializations. Table 6.2 shows a summary of the results and table B.19 shows the raw data. The more hidden layers were used, the more have the results varied.

System	Classification error								
	TOP1			TOP3			MER		
	min	max	range	min	max	range	min	max	range
B_1	23.08 %	23.44 %	0.36 %	6.67 %	6.80 %	0.13 %	6.54 %	6.64 %	0.10 %
B_2	21.45 %	21.83 %	0.38 %	5.68 %	5.75 %	0.07 %	5.60 %	5.68 %	0.08 %
B_3	21.54 %	22.28 %	0.74 %	5.50 %	5.82 %	0.32 %	5.41 %	5.75 %	0.34 %
B_4	23.19 %	24.84 %	1.65 %	5.98 %	6.44 %	0.46 %	5.83 %	6.21 %	0.38 %

Table 6.2.: *The systems B_1 – B_4 were randomly initialized, trained and evaluated 5 times to estimate the influence of random weight initialization.*

6.2. Preprocessing Algorithms

The preprocessing algorithms can be split in two groups as shown in section 3.2: Normalization and noise reduction algorithms.

Both, normalization and noise reduction algorithms, can be analyzed for computational costs and effect on the test classification error. Additionally, noise reduction algorithms can be analyzed for effectiveness in terms of false positives or false negatives. However, in the following they were only analyzed for their effect on the three error measures TOP1, TOP3 and MER.

6.2.1. Scale and Shift

There are several ways to implement the scale and shift algorithm. Especially how one deals with dots or straight lines ($x_{\text{max}} - x_{\text{min}} = 0$ or $y_{\text{max}} - y_{\text{min}} = 0$) makes a difference.

The following transformation is done with each point:

$$\begin{aligned} p['x'] &\leftarrow (p['x'] - x_{\text{min}}) \cdot factor - add_x \\ p['y'] &\leftarrow (p['y'] - y_{\text{min}}) \cdot factor - add_y \\ p['t'] &\leftarrow (p['t'] - t_{\text{min}}) \end{aligned}$$

where $x_{\min}$, $y_{\min}$ and $t_{\min}$ are the minimal values among all points of a single recording, $factor \in \mathbb{R}^+$ is positive scaling constant and $add_x, add_y \in \mathbb{R}_0^+$ are non-negative shifting constants.

Implementation 1 is the implementation that was used for all other evaluations. It does not shift the bigger dimension, but centers the smaller dimension of the bounding box within the $[-1, 1] \times [-1, 1]$ unit square.

A recording with a bounding box of the dimension 1×0.8 would be within $[-0.4, 0.4] \times [0.0, 1.0]$ after implementation 1 shifting.

The following lines show how such a shifting could be implemented:

```
$width \leftarrow x_{\max} - x_{\min}$
$height \leftarrow y_{\max} - y_{\min}$
$factor_x, factor_y \leftarrow 1, 1$
**if** $width \neq 0$ **then**
    $factor_x \leftarrow \frac{1}{width}$
**if** $height \neq 0$ **then**
    $factor_y \leftarrow \frac{1}{height}$
$factor \leftarrow \textsc{min}(factor_x, factor_y)$
$add_x, add_y \leftarrow 0, 0$
$\triangleright$ Only the smaller dimension ($x$ or $y$) gets centered
$add \leftarrow -\frac{\textsc{min}(width,height) \cdot factor}{2}$
**if** $factor == factor_x$ **then**
    $add_y \leftarrow add$
**else**
    $add_x \leftarrow add$
```

Implementation 2 is the same as implementation 1, but with $add_x = 0$ and $add_y = 0$. So no centering was done. After that, the recording is in the $[0, 1] \times [0, 1]$ unit square, aligned to $(0, 0)$.

A recording with a bounding box of the dimension 1×0.8 would be within $[0.0, 0.8] \times [0.0, 1.0]$ after implementation 2 shifting.

Implementation 3 is the same as implementation 1, but with the bigger dimension being shifted by -0.5. So in implementation 1, only one dimension gets centered around $(0, 0)$. In implementation 3, both dimensions get centered around $(0, 0)$.

```
**if** $factor == factor_x$ **then**
    $add_y \leftarrow add$
    $add_x \leftarrow -0.5$
**else**
    $add_x \leftarrow add$
    $add_y \leftarrow -0.5$
```

A recording with a bounding box of the dimension 1×0.8 would be within $[-0.4, 0.4] \times [-0.5, 0.5]$ after implementation 3 shifting.

Those three implementations of the scale and shift algorithm were tested with all the neural networks B_1–B_4. The results in table B.20 show that system B_4 was most sensitive for changes in this implementation. Implementation 2 performed best or was at lest not more than 0.04 % worse than implementation 1 for B_1–B_3. However, implementation 2 was by far the worst for B_4. The experiment was executed four times with different weight initializations for $B_{4,I2}$ and all evaluations were at least 3 % worse in TOP1 error than B_4.

6.2.2. Wild Point Filter

Wild points are strokes which consist of a single point which the user did not want to draw. Wild points are likely to be caused by hardware errors (see problem D1, page 10).

The dataset contained 2.77 % recordings with dots, excluding all recordings of the symbols i, j, `\cdot`, `\div`, `\because` and `\therefore`. However, removing those dots changed the bounding box size of only 0.85 % of all recordings.

As the proposed wild point detection relies only on the speed of single points of a stroke it was analyzed in which range those points are. The mean speed was $0.35\,\frac{\text{px}}{\text{ms}}$ with a standard deviation of $0.65\,\frac{\text{px}}{\text{ms}}$. Figure 6.1 shows the distribution of the speed between control points in a histogram. After that, the wild point filter with a threshold $\theta = 3\,\frac{\text{px}}{\text{ms}}$ and $\theta = 6\,\frac{\text{px}}{\text{ms}}$ and were tested. The results are listed in table 6.3. The models B_3 and B_4 improved by both applications, whereas the models B_1 and B_2 did not improve.

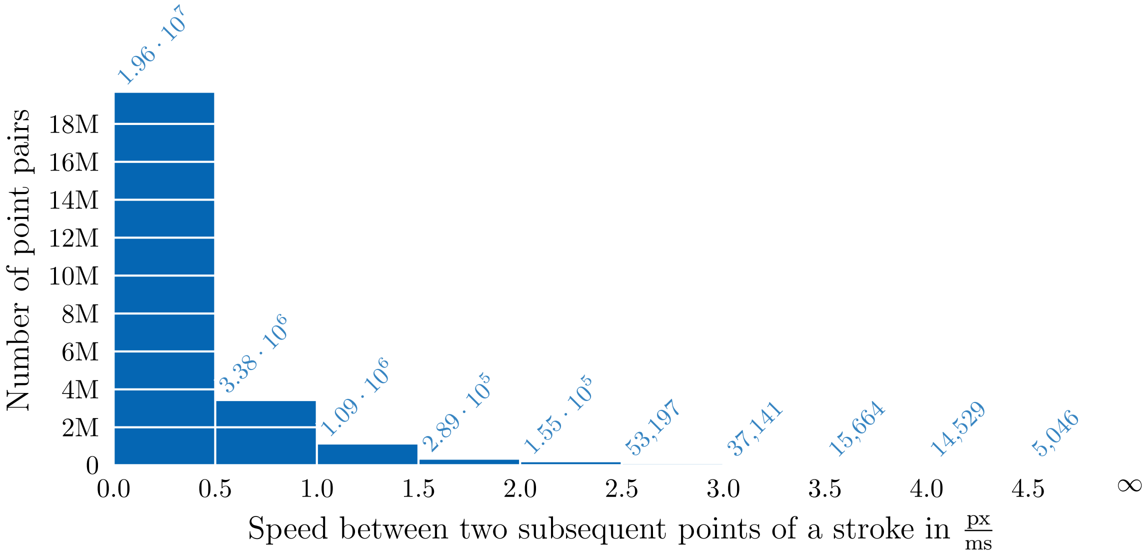


Figure 6.1.: *The speed between two subsequent control points of the same stroke is used can be used for point filtering preprocessing steps. Points with high speeds could be caused by errors in the hardware. The plot shows that the majority of all point pairs have a speed of less than* $0.5\,\frac{\text{px}}{\text{ms}}$*. Less than* 0.3 % *of all point pairs have a speed of more than* $3\,\frac{\text{px}}{\text{ms}}$*.*

6.2.3. Stroke Connect

In order to solve problem D5 (interrupted strokes, see page 10) the stroke connect algorithm was introduced on page 15. The idea is that for a pair of consecutively drawn strokes s_i, s_{i+1} the last point s_i is close to the first point of s_{i+1} if a stroke was accidentally split into two strokes.

Figure 6.2 shows the distance between consecutively drawn stroke pairs. 59 % of all stroke pair distances are between 30 px and 150 px. Hence the stroke connect algorithm was tried with 5 px, 10 px and 20 px. Table B.21 shows the results of this algorithm. All models improved much with a threshold of $\theta = 10\,\text{px}$ with all error measures, except B_4 with the TOP3 error measure.

System	Classification error					
	TOP1	change	TOP3	change	MER	change
$B_{1,\theta_w=3\frac{\text{px}}{\text{ms}}}$	23.55 %	0.21 %	6.85 %	0.05 %	6.70 %	0.06 %
$B_{2,\theta_w=3\frac{\text{px}}{\text{ms}}}$	21.73 %	0.22 %	5.67 %	−0.08 %	5.58 %	−0.09 %
$B_{3,\theta_w=3\frac{\text{px}}{\text{ms}}}$	21.25 %	−0.68 %	5.66 %	−0.08 %	5.55 %	−0.09 %
$B_{4,\theta_w=3\frac{\text{px}}{\text{ms}}}$	21.91 %	−1.97 %	5.77 %	−0.35 %	5.65 %	−0.39 %
$B_{1,\theta_w=6\frac{\text{px}}{\text{ms}}}$	23.30 %	−0.04 %	6.94 %	0.14 %	6.80 %	0.16 %
$B_{2,\theta_w=6\frac{\text{px}}{\text{ms}}}$	21.80 %	0.29 %	5.77 %	0.02 %	5.65 %	−0.02 %
$B_{3,\theta_w=6\frac{\text{px}}{\text{ms}}}$	22.30 %	−0.63 %	5.79 %	0.05 %	5.58 %	−0.06 %
$B_{4,\theta_w=6\frac{\text{px}}{\text{ms}}}$	22.98 %	−0.90 %	6.10 %	−0.02 %	5.99 %	−0.05 %

Table 6.3.: *The wild point filtering algorithm was added to the baseline systems B_1–B_4 and the absolute change to the baseline model of the same topology was calculated. The threshold θ_w, which is measured in $\frac{\text{px}}{\text{ms}}$, denotes which points get filtered. All points that are faster than the threshold get filtered. This means a threshold $\theta = 0\,\frac{\text{px}}{\text{ms}}$ results in the same as if no wildpoint filter had been applied. The table shows that the effect of wild point filtering is less than the effect of random weight initialization for the models B_1–B_3, but improves model B_4.*

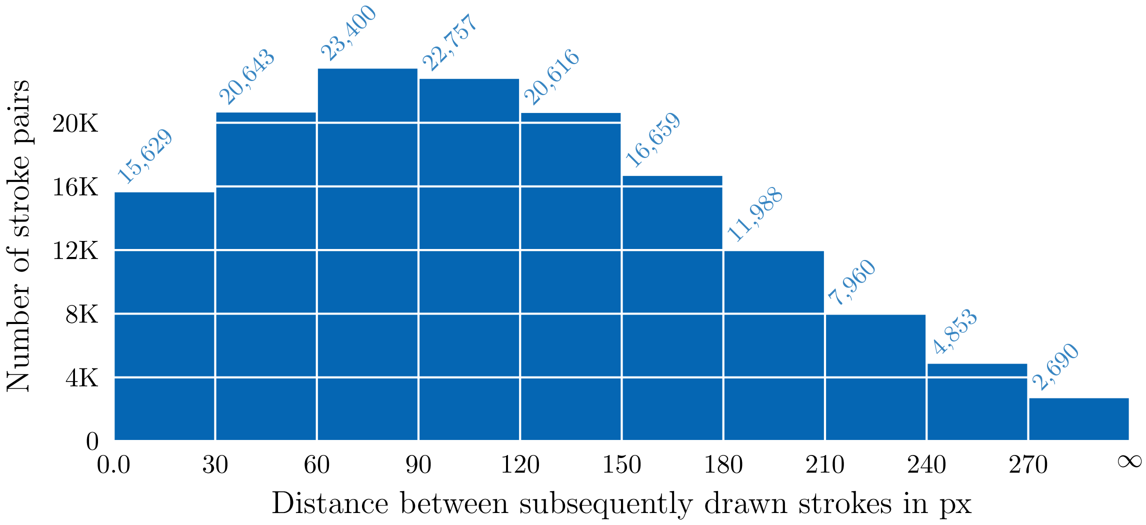


Figure 6.2.: *The distance between two subsequently drawn strokes (s_i, s_{i+1}) in pixels is calculated by measuring the euclidean distance between the last point of s_i and the first point of s_{i+1}. Less than 11 % of those distances are below 30 px. It is assumed that accidentally interrupted strokes (see page 10) are rarely happening. The stroke connect algorithm is therefore evaluated with thresholds of less than 30 px.*

6.2.4. Weighted Average Smoothing

Weighted average smoothing was described in section 3.2.2 on page 14. It takes consecutive points, weights the x, y and $time$ values independently and calculates a new average point. Three points were used to calculate the new average point with weights $w_1 = [\frac{1}{6}, \frac{4}{6}, \frac{1}{6}]$ and $w_2 = [\frac{1}{3}, \frac{1}{3}, \frac{1}{3}]$. The results are shown in table 6.4. The results with w_1 did not change enough to make a meaningful statement about the influence of this algorithm, but w_2 had a positive effect on $B_1 - B_3$.

System	Weights	Classification error					
		TOP1	change	TOP3	change	MER	change
$B_{1,WAS}$	$\frac{1}{6}, \frac{4}{6}, \frac{1}{6}$	23.33 %	−0.01 %	6.68 %	−0.12 %	6.57 %	−0.07 %
$B_{2,WAS}$	$\frac{1}{6}, \frac{4}{6}, \frac{1}{6}$	21.73 %	0.22 %	5.87 %	0.12 %	5.75 %	0.08 %
$B_{3,WAS}$	$\frac{1}{6}, \frac{4}{6}, \frac{1}{6}$	21.77 %	−0.16 %	5.57 %	−0.17 %	5.52 %	−0.12 %
$B_{4,WAS}$	$\frac{1}{6}, \frac{4}{6}, \frac{1}{6}$	24.17 %	0.29 %	6.47 %	0.35 %	6.21 %	0.17 %
$B_{1,WAS}$	$\frac{1}{3}, \frac{1}{3}, \frac{1}{3}$	23.26 %	−0.08 %	6.55 %	−0.25 %	6.41 %	−0.23 %
$B_{2,WAS}$	$\frac{1}{3}, \frac{1}{3}, \frac{1}{3}$	21.67 %	−0.16 %	5.69 %	−0.06 %	5.60 %	−0.07 %
$B_{3,WAS}$	$\frac{1}{3}, \frac{1}{3}, \frac{1}{3}$	21.44 %	−0.49 %	5.67 %	−0.07 %	5.58 %	−0.06 %
$B_{4,WAS}$	$\frac{1}{3}, \frac{1}{3}, \frac{1}{3}$	24.24 %	0.36 %	6.26 %	0.14 %	5.95 %	−0.09 %

Table 6.4.: *The baseline models B_1–B_4 were tested with additionally weighted average smoothing (WAS) being applied. The smoothing was applied before every other preprocessing step.*

6.2.5. Douglas-Peucker Smoothing

The Douglas-Peucker algorithm, which is described on page 15, can be used to find control points that are more relevant for the overall shape of a recording. After that, an interpolation can be done. If the interpolation is a cubic spline interpolation, this makes the recording smooth.

The Douglas-Peucker algorithm was applied with a threshold of $\varepsilon = 0.05$, $\varepsilon = 0.1$ and $\varepsilon = 0.2$ after scaling and shifting, but before the interpolation. The interpolation was done linearly and with cubic splines in two experiments. The recording was scaled and shifted again after the interpolation because the bounding box might have changed.

The result of the application of the Douglas-Peucker smoothing with $\varepsilon > 0.05$ was a high rise of all classification error measures for all models. This means that the simplification process removes some relevant information and does not — as it was expected — remove only noise. For $\varepsilon = 0.05$ with linear interpolation some models improved for some error measures, but the changes were small. It could be an effect of random weight initialization. However, cubic spline interpolation made all systems perform much worse.

The lower the value of ε, the less does the recording change after this preprocessing step. As it was applied after scaling the recording such that the biggest dimension of the recording (width or height) is 1, a value of $\varepsilon = 0.05$ means that a point has to move at least 5 percent of the biggest dimension.

Table B.22 shows the evaluation results.

6.3. Data Multiplication

Data multiplication can be used to make the model invariant to transformations. However, this idea seems not to work well in the domain of on-line handwritten mathematical symbols. It was tried to triple the data by adding a rotated version that is rotated 3 degrees to the left and another one that is rotated 3 degrees to the right around the center of mass. This data multiplication made all classifiers for most error measures perform worse than before as table 6.5 shows.

Data multiplication was also used in section 6.6.6 on page 51 combined with newbob training.

System	Classification error					
	TOP1	change	TOP3	change	MER	change
$B_{1,min=-3,max=3,num=3}$	25.43 %	2.09 %	6.44 %	−0.36 %	6.28 %	−0.36 %
$B_{2,min=-3,max=3,num=3}$	<u>23.58 %</u>	2.07 %	<u>5.78 %</u>	0.03 %	<u>5.57 %</u>	−0.10 %
$B_{3,min=-3,max=3,num=3}$	28.90 %	6.97 %	8.39 %	2.65 %	7.54 %	1.90 %
$B_{4,min=-3,max=3,num=3}$	39.71 %	15.83 %	18.12 %	12.00 %	15.20 %	9.16 %

Table 6.5.: *Evaluation of the baseline models that used a training set T' which was three times bigger than the normal training set T. T' was created from T by adding two rotational variants for each original recording. Those two rotational variants were rotated by $-3°$ and by $3°$ around the center of mass.*

6.4. Features

A single dimension of a feature F of a given symbol S could be modeled by a random variable. For a random variable X_S that is normally distributed and has a mean of μ for the feature F and a standard deviation of σ one writes:

$$X_{S,F} \sim \mathcal{N}(\mu, \sigma^2)$$

In the interval $(\mu_S - 2\sigma_S, \mu + 2\sigma)$ is about 99.7 % of the data. That means if those intervals are disjunct for two given symbols, the symbols can be separated well by the feature F. This knowledge can be used to calculate the mean μ and the standard deviation σ of every symbol for a given feature. The symbol can then be plotted in a mean-standard deviation scatter plot at the coordinates (μ, σ). Ideally, the intra-symbol standard deviation would be low, the inter-symbol standard deviation would be high and the means of the symbols would be well separated from each other.

For example, in figure 6.3 one can see that the symbol $-$ at $(0.04, 0.03)$ can be distinguished from many other symbol only by using the re-curvature feature for the first stroke. In contrast, the $\top$ symbol cannot be distinguished from any other symbol by this feature.

Five features are evaluated in the following. The mean and variance of the first dimension of those features was plotted to give the reader an impression of how well they separate symbols and which symbols cannot be separated by those single features. Additionally, the baseline systems were extended by those features to measure their influence on the three error measures.

6.4.1. Re-curvature

The re-curvature feature is a feature for single strokes. It was defined on page 19 as

$$\text{re-curvature}(stroke) := \frac{\text{height}(stroke)}{\text{length}(stroke)}$$

As both, the height and the distance are measured in the same unit, the feature is a dimensionless quantity.

In order to get a constant feature dimension it is required to define on how many strokes this feature should get applied to. If a recording has less strokes, the feature is defined to have the value 0.

The results of this feature, applied to the first four strokes, are shown in table 6.6. This feature improved classification for all models and all error measures a lot.

Figure 6.3 shows the mean and standard deviation of the re-curvature feature for the first stroke of every recording.

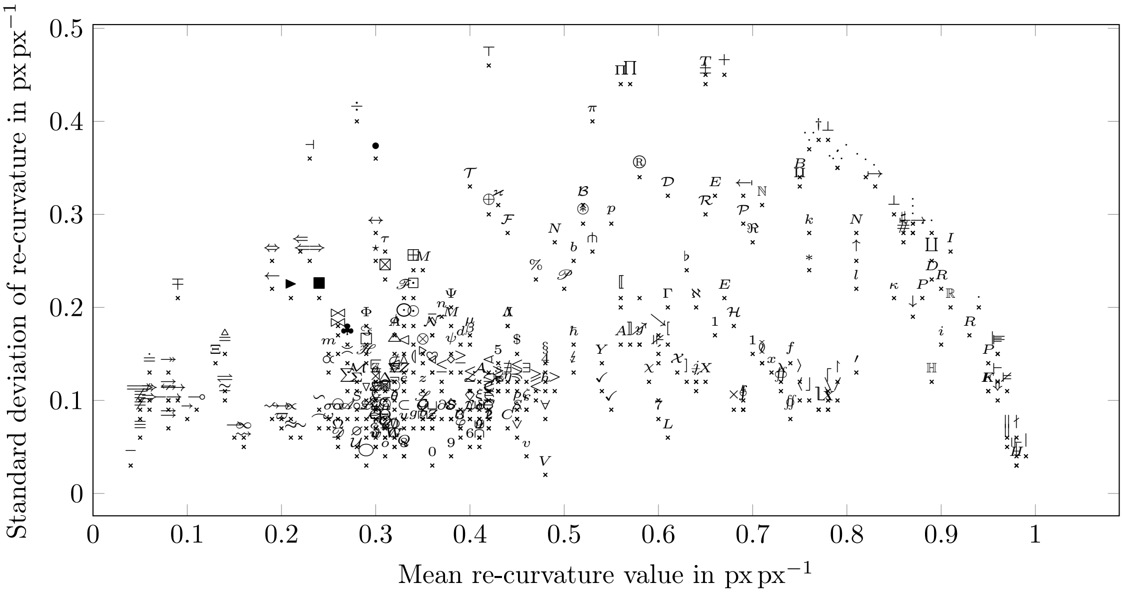


Figure 6.3.: *Mean and standard deviation of the re-curvature feature of the first stroke every symbol. The – (0.04,0.03) and the ⇝ (0.16,0.05) are well-separated, but this feature is not able to distinguish ± (0.65,0.44) from either of them.*

System	Classification error					
	TOP1	change	TOP3	change	MER	change
$B_{1,rec}$	22.14 %	−1.20 %	5.96 %	−0.84 %	5.86 %	−0.78 %
$B_{2,rec}$	20.65 %	−0.86 %	5.19 %	−0.56 %	5.10 %	−0.57 %
$B_{3,rec}$	20.84 %	−1.09 %	5.30 %	−0.44 %	5.22 %	−0.42 %
$B_{4,rec}$	23.25 %	−0.63 %	5.90 %	−0.22 %	5.65 %	−0.39 %

Table 6.6.: *Evaluation of baseline systems B_1 – B_4 with an additional re-curvature feature (rec) for each of the 4 strokes. All error measures improved notably.*

6.4.2. Stroke Center Point

The stroke center point is a 2-dimensional feature. It calculates the center of mass of a stroke by calculating the arithmetic mean of its coordinates. The feature was added to all four baseline systems B_1 – B_4. As those systems had four strokes, the feature was applied for four strokes resulting in 8 new features.

Table 6.7 shows the results of this experiment. The results changed by less than the range of random weight initialization which indicates that this feature is useless.

System	Classification error					
	TOP1	change	TOP3	change	MER	change
$B_{1,cp}$	23.18 %	−0.16 %	6.53 %	−0.27 %	6.39 %	−0.25 %
$B_{2,cp}$	21.74 %	0.23 %	5.85 %	0.10 %	5.74 %	0.07 %
$B_{3,cp}$	21.19 %	−0.74 %	5.63 %	−0.11 %	5.55 %	−0.09 %
$B_{4,cp}$	23.94 %	0.06 %	6.24 %	0.12 %	6.08 %	0.04 %

Table 6.7.: *Evaluation of baseline systems B_1 – B_4 with additional stroke center point features (cp) for 4 strokes. The results indicate that the feature is useless.*

6.4.3. Ink

The ink feature measures how long each stroke is. This feature improved all models except for B_4 as one can see in the results listed in table 6.8. Although the experiment was executed multiple times for B_4, all evaluations showed that the ink feature made B_4 perform worse.

The mean-standard deviation scatterplot is shown in figure C.1

System	Classification error					
	TOP1	change	TOP3	change	MER	change
$B_{1,i}$	22.22 %	−1.12 %	5.88 %	−0.92 %	5.77 %	−0.87 %
$B_{2,i}$	20.91 %	−0.60 %	5.20 %	−0.55 %	5.11 %	−0.53 %
$B_{3,i}$	21.33 %	−0.60 %	5.28 %	−0.46 %	5.21 %	−0.43 %
$B_{4,i}$	27.15 %	3.27 %	6.60 %	0.48 %	6.31 %	0.27 %

Table 6.8.: *Evaluation of baseline systems B_1 – B_4 with additional ink feature (i). The small systems B_1 – B_3 benefit from this feature, but the bigger model B_4 performs worse.*

6.4.4. Stroke Count

The number of strokes is a strong single feature, because most people tend to use the same number of strokes for a given symbol. There are some symbols where people make variations, but those variations are only if two strokes are connected or not. For example, The letter “E” is by some people drawn as 4 strokes, by others as an “L” with two more strokes. Figure 6.4 shows the mean-standard deviation scatterplot of the number of strokes for each of the evaluated symbols excluding a few listed in table B.8.

The feature improves recognition rates for the models B_1–B_3, but makes B_4 perform worse.

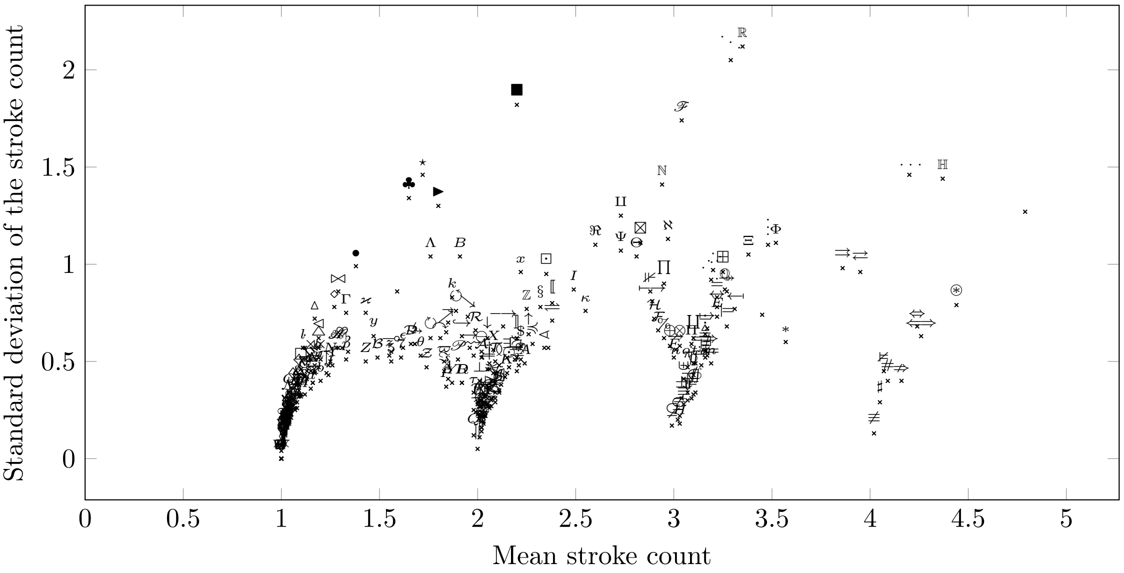


Figure 6.4.: *Mean-standard deviation scatterplot of the stroke count feature as it was introduced on page 43.*

System	Classification error					
	TOP1	change	TOP3	change	MER	change
$B_{1,sc}$	23.28 %	−0.06 %	6.71 %	−0.09 %	6.55 %	−0.09 %
$B_{2,sc}$	21.75 %	−0.24 %	5.46 %	−0.29 %	5.38 %	−0.29 %
$B_{3,sc}$	21.42 %	−0.51 %	5.45 %	−0.29 %	5.39 %	−0.25 %
$B_{4,sc}$	25.28 %	1.40 %	6.87 %	0.75 %	6.50 %	0.46 %

Table 6.9.: *Evaluation of baseline systems B_1 – B_4 with additional stroke count feature (sc). $B_{2,sc}$ and $B_{3,sc}$ performed slightly better, $B_{1,sc}$ barely changed, but $B_{4,sc}$ performed much worse than before.*

6.4.5. Aspect Ratio

The aspect ratio is a 1 dimensional feature of a recording that is calculated by calculating the ratio

$$\text{aspect\_ratio}(recording) = \frac{\text{width}(recording)}{\text{height}(recording)}$$

However, as the width (and the height) get calculated by subtracting the minimum x (y) value from the maximum x (y) value, it can be 0. In order to avoid zero division errors

+0.01 was added to both values, width and height. This could be seen as the thickness of a stroke and could have an impact on the value of this feature.

One dimension — either width or height — has the value 1 as all baseline systems use the scale and shift algorithm (except if it was only a point).

Figure C.2 on page 85 shows a scatterplot of the mean and the standard deviation of this feature and table B.9 on page 76 lists all symbols that were not used in the figure.

The evaluation results showed that the models B_1 and B_3 improved with this feature by about 0.2 % MER error, but B_4 got worse. The error rate of model $B_{2,ar}$ barely changed.

System	Classification error					
	TOP1	change	TOP3	change	MER	change
$B_{1,ar}$	23.07 %	−0.27 %	6.55 %	−0.25 %	6.45 %	−0.19 %
$B_{2,ar}$	21.45 %	−0.06 %	5.67 %	−0.08 %	5.60 %	−0.07 %
$B_{3,ar}$	21.49 %	−0.44 %	5.36 %	−0.38 %	5.28 %	−0.36 %
$B_{4,ar}$	25.01 %	1.13 %	6.45 %	0.33 %	6.10 %	0.06 %

Table 6.10.: *Evaluation of baseline systems B_1 – B_4 with additional aspect ratio feature (ar). The small systems $B_{1,ar}$–$B_{3,ar}$ improved, but $B_{4,ar}$ got worse.*

6.5. System A: Greedy Time Warping

System A used only scaling and shifting as resampling. After that, greedy time warping was applied (a variant of dynamic time warping that is faster, but not optimal) to calculate the distance of two recordings.

A new recording was classified getting the minimal distance to any known recording. The label of the recording with minimal distance was used as a classification result.

This system needed about 22 s in average on Intel Pentium P6200 processor to classify a single recording, although the amount of recordings per symbol was limited to 50 at maximum. So it was much too slow.

The classification error was 85.46 %. The TOP10 classification error was 65.78 %. So this system was clearly much worse than the MLPs.

Note that those results are much worse than what was achieved in [Kir10] on a similar dataset. This is probably the reason, because Kirsch did apply more preprocessing steps and tweaked the time warping approach. However, even his results with time warping were much worse than what can be done with MLPs.

6.6. System B: Multilayer Perceptrons

The tested systems were already described in table 6.1. However, there are many parameters that might influence how fast a MLP can learn. The effect of changes to some of those parameters were tested and are described in the following.

6.6.1. Baseline Testing

Figure 6.5, a plot of the validation and test error over the epochs shows that — except for the system B_4 — the biggest drops in error are done until the 400th epoch. After that, there is almost no change. Only system B_4 seems to be able to improve after that.

System	Classification error					
	TOP1	change	TOP3	change	MER	change
$B_{1,epochs=10\,000}$	21.31 %	−2.03 %	5.81 %	−0.99 %	5.68 %	−0.96 %
$B_{2,epochs=10\,000}$	21.47 %	−0.04 %	5.84 %	0.09 %	5.68 %	0.01 %
$B_{3,epochs=10\,000}$	21.16 %	−0.77 %	$\underline{5.44\,\%}$	−0.30 %	$\underline{5.34\,\%}$	−0.30 %
$B_{4,epochs=10\,000}$	$\underline{21.00\,\%}$	−2.88 %	5.74 %	−0.38 %	5.64 %	−0.40 %

Table 6.11.: *Evaluation of baseline systems B_1 – B_4 after* 10 000 *epochs of training. The models were tested after every epoch. There was no model in between that performed much better. So overfitting is not a problem in that case.*

Training B_4 for 10 000 epochs led to an TOP1 error of 21.00 % which is 2.88 % better than the result with only 1000 epochs of training. Model B_3 was also able to improve, but not that much. The exact results for all models and errors are in table 6.11.

Figure 6.6 shows a learning curve for model B_1. Every point in that plot was generated by artificially reducing the training set to a maximum number (from 1 to 150) of training examples per symbol and training for only 300 epochs. As the training and the test error were plotted, one can see how more data might affect the error rate. Although there are symbols with only 50 recordings, it seems not to make a difference if one had 150 or more recordings per symbol for a MLP with only one hidden layer. The TOP1 error on the training set for 6 training examples per symbol is already at 29 %. This indicates that more or better features could improve the model, whereas more training examples will not help to get below 29 %.

6.6.2. Execution Time

The neural network training was executed on a Nvidia GeForce GTX Titan Black. It took about 12 minutes to train B_1 and about 25 minutes to train B_4.

The training of B_1 executed on a Intel P6200 CPU was aborted after 31 h. This means the execution time was reduced by GPU training to about 1.6 % of the time it took before.

6.6.3. Learning Rate

The choice of the learning rate in mini batch training determines how wide the steps in gradient descent are. Bigger steps lead to a faster improvement at the beginning, but at the end the algorithm might jump back and forth and not be able to improve. Table B.23 shows the results after 1000 epochs of mini batch training with different choices for the learning rate. Tested were learning rates of 0.05, 0.1, 0.2 and 1.0. For all models, a learning rate of 0.1 was the best choice.

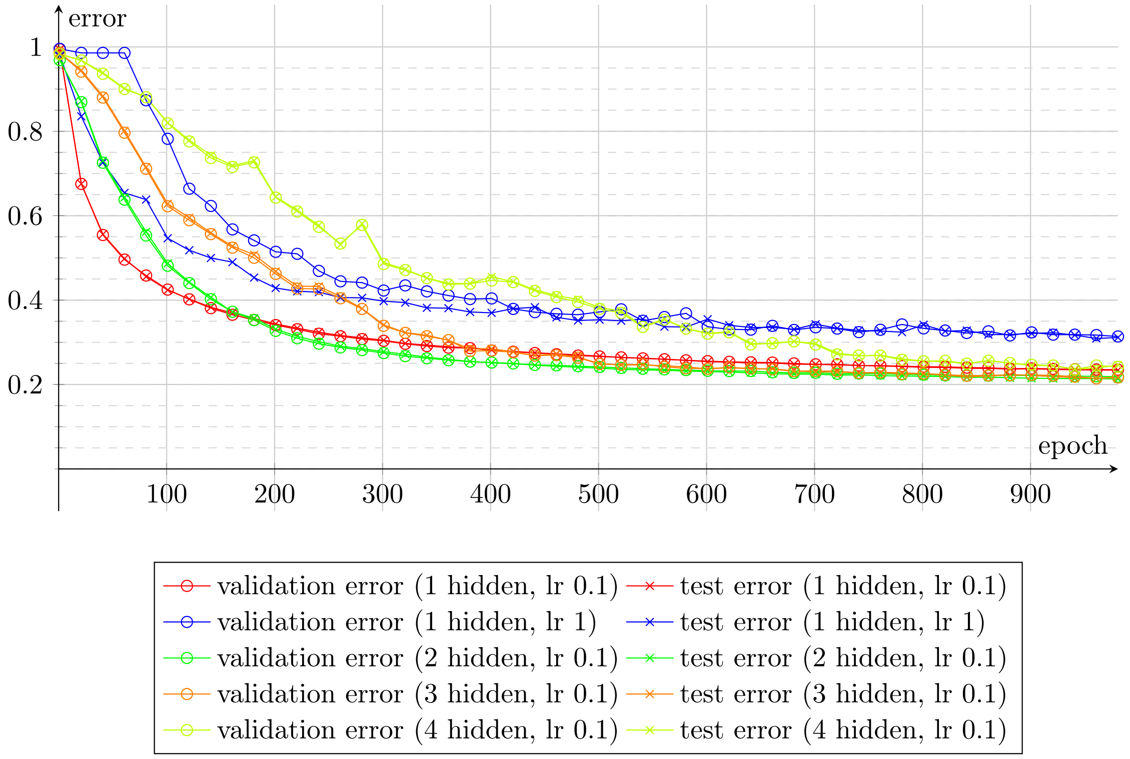


Figure 6.5.: *Training- and test error by number of trained epochs for different topologies. The curves all show almost no difference in validation and test error. The error for all curves with learning rates of $\eta = 0.1$ converge to similar values, although the error of B_4 still drops while all other models do not perform better with more training. A learning rate of $\eta = 1$ is much worse than $\eta = 0.1$.*

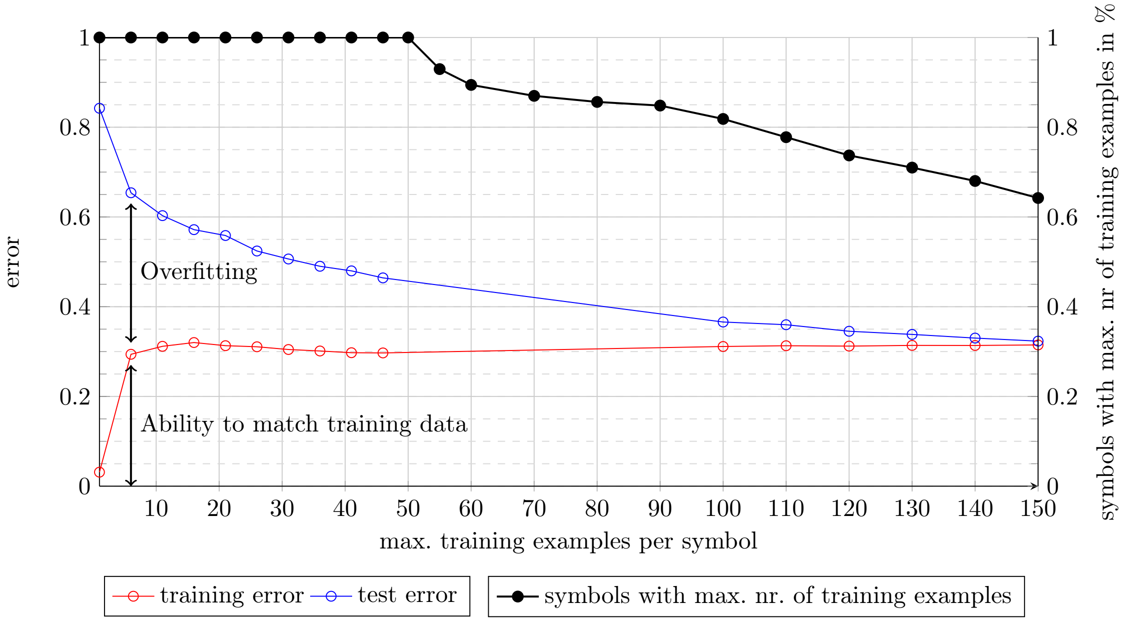


Figure 6.6.: *This plot shows the learning curve for a 160:500:369 MLP. The x axis shows the number of training examples per symbol, the y axis shows the error for the colored lines and the percentage of symbols that had at least the maximum number of training examples. The training and the testing error is plotted as well as the percentage of symbols with the maximum number of training examples.*

6.6.4. Momentum

The momentum α in mini-batch training is, just like the learning rate, important for the speed of improvements and eventually also for the final result. It was explained in section 4.4.6.

System	Classification error					
	TOP1	change	TOP3	change	MER	change
$B_{1,\alpha=0.1}$	23.34 %		6.80 %		6.64 %	
$B_{1,\alpha=0.9}$	22.51 %	−0.83 %	6.88 %	0.08 %	6.74 %	0.10 %
$B_{2,\alpha=0.1}$	21.51 %		5.75 %		5.67 %	
$B_{2,\alpha=0.9}$	21.17 %	−0.34 %	5.96 %	0.21 %	5.85 %	0.18 %
$B_{3,\alpha=0.1}$	21.93 %		5.74 %		5.64 %	
$B_{3,\alpha=0.9}$	20.57 %	−1.36 %	5.59 %	−0.15 %	5.53 %	−0.11 %
$B_{4,\alpha=0.1}$	23.88 %		6.12 %		6.04 %	
$B_{4,\alpha=0.9}$	21.39 %	−2.49 %	6.00 %	−0.12 %	5.91 %	−0.13 %

Table 6.12.: *Evaluation results of the systems B_1 – B_4 with adjusted momentums α. The column "change" was left blank, because the baseline systems B_1 – B_4 use a momentum of $\alpha = 0.1$.*

6.6.5. Pretraining

Pretraining is a technique used to improve the training of deep neural networks. Two pretraining algorithms are described in section 4.4.9 on page 29: SLP and denoising auto-encoders.

Figure 6.7 shows the evolution of validation and test errors over 1000 epochs with supervised layer-wise pretraining and without pretraining. It clearly shows that this kind of pretraining improves the classification performance. Detailed results are listed in table 6.13.

System	Classification error					
	TOP1	change	TOP3	change	MER	change
B_1	23.34 %		6.80 %		6.64 %	
$B_{2,SLP}$	19.89 %	−1.62 %	4.76 %	−0.99 %	4.68 %	−0.99 %
$B_{3,SLP}$	19.43 %	−2.50 %	4.64 %	−1.10 %	4.54 %	−1.10 %
$B_{4,SLP}$	19.63 %	−4.25 %	4.66 %	−1.46 %	4.55 %	−1.49 %

Table 6.13.: *Systems with SLP compared to pure gradient descent. The SLP systems performed notably better. Although the pretrained systems $B_{i,SLP}$ got* 1000 *epochs of training for each layer whereas the systems B_i only got* 1000 *epochs of training in total, it is important to note that the systems B_1–B_3 were not able to improve with more training epochs. Denoising auto-encoders (da) on the other hand made the system much worse.*

Pretraining with denoising auto-encoder lead to the much worse results listed in table 6.14. The first layer used a tanh activation function. Every layer was trained for 1000 epochs and the MSE loss function. A learning-rate of $\eta = 0.001$, a corruption of 0.3 and a L_2 regularization of $\lambda = 10^{-4}$ were chosen. This pretraining setup made all systems with all error measures perform much worse.

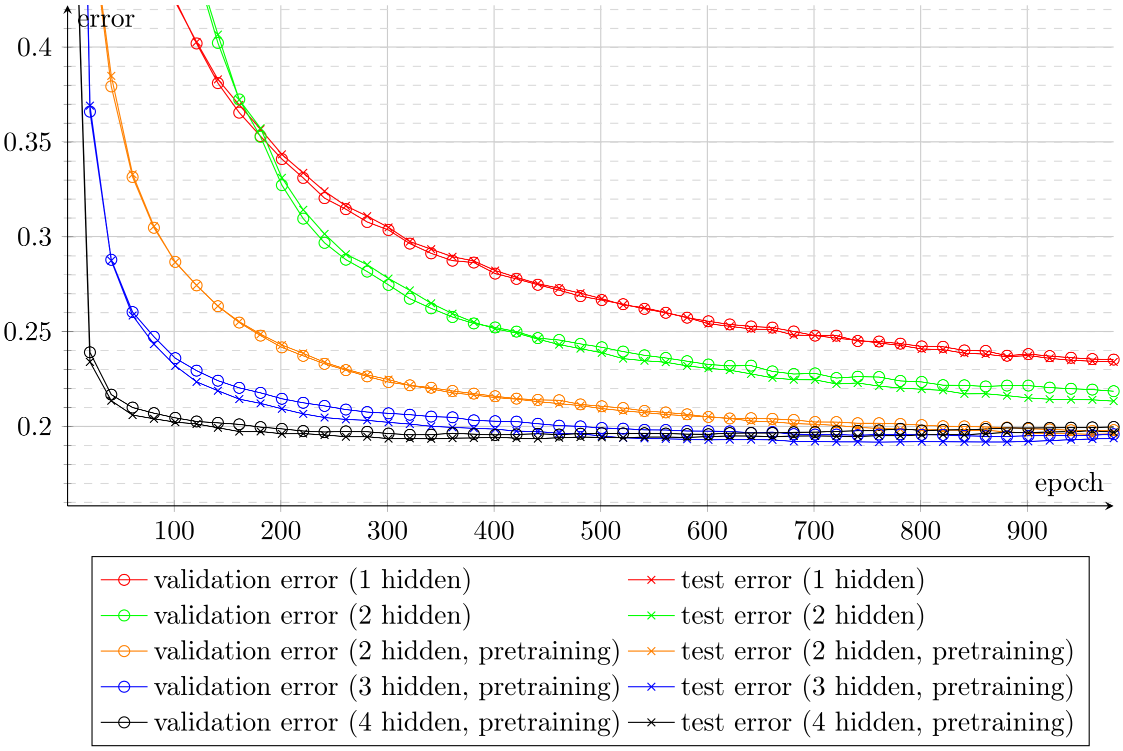


Figure 6.7.: *Training- and test error by number of trained epochs for different topologies with SLP. The plot shows that all pretrained systems performed much better than the systems without pretraining. All plotted systems did not improve with more epochs of training.*

6.6.6. Newbob Training

Newbob is a training mode that adjusts the training rate according to the improvement over the last epoch. It was explained on page 28. Figure 6.8 shows the result of the training and test error with newbob training. It was used with $\theta_1 = 0.5\,\%$ and $\theta_2 = 0.1\,\%$.

Table 6.15 shows results for different choices of parameters for newbob training. Although the training finished fast (most of the time after about 80 epochs, but always before the 400th epoch), no result of newbob training was better than the baseline system.

System	Classification error					
	TOP1	change	TOP3	change	MER	change
$B_{1,p}$	23.75 %	0.41 %	7.19 %	0.39 %	6.98 %	0.34 %
$B_{2,p}$	22.76 %	1.25 %	6.38 %	0.63 %	6.28 %	0.61 %
$B_{3,p}$	23.10 %	1.17 %	6.14 %	0.40 %	6.04 %	0.40 %
$B_{4,p}$	25.59 %	1.71 %	6.99 %	0.87 %	6.88 %	0.84 %

Table 6.14.: *Systems with denoising auto-encoder pretraining compared to pure gradient descent. The pretrained systems clearly performed worse.*

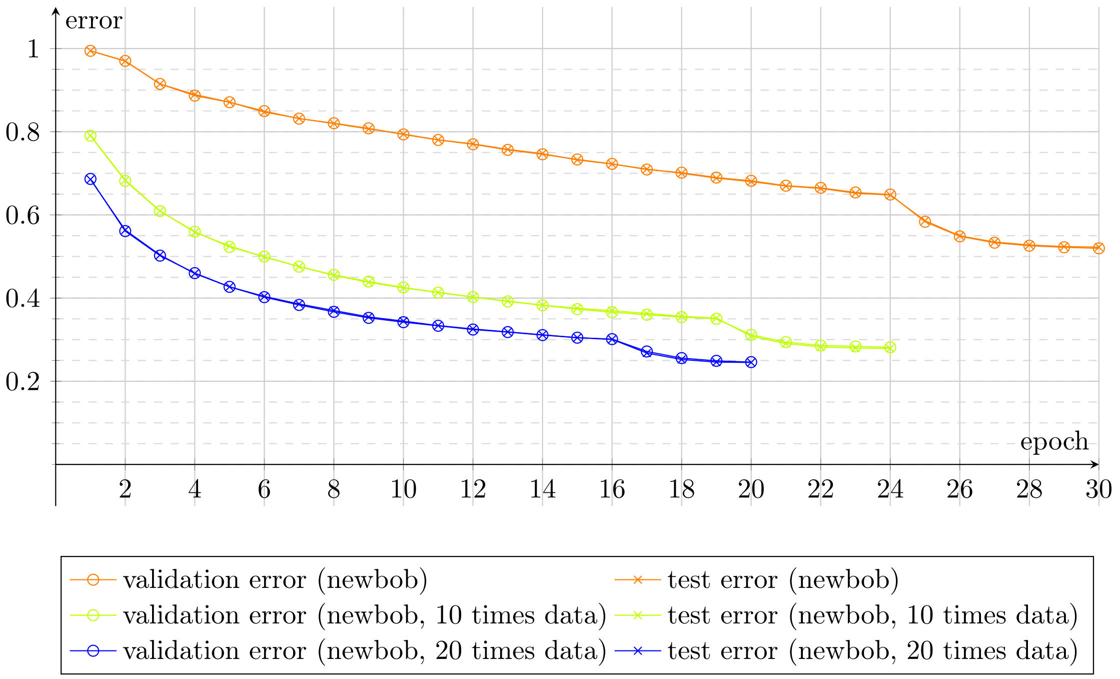


Figure 6.8.: *Training- and test error by number of trained epochs for different topologies and training types.*

System	Classification error					
	TOP1	change	TOP3	change	MER	change
$B_{1,\alpha=0.1,dm=20,d=0.5,\theta=0.001}$	34.86 %	11.52 %	16.02 %	9.22 %	15.69 %	9.05 %
$B_{1,\alpha=1.0,dm=20,d=0.95,\theta=0.00001}$	30.60 %	7.26 %	12.36 %	5.56 %	12.09 %	5.45 %
$B_{1,\alpha=0.1,dm=20,d=0.50,\theta=0.00001}$	28.24 %	4.90 %	10.92 %	4.12 %	10.68 %	4.04 %
$B_{1,\alpha=0.1,dm=20,d=0.75,\theta=0.00001}$	27.70 %	4.36 %	10.67 %	3.87 %	10.41 %	3.77 %
$B_{1,\alpha=0.1,dm=20,d=0.95,\theta=0.00001}$	26.77 %	3.43 %	9.59 %	2.79 %	9.42 %	2.78 %
$B_{1,\alpha=0.1,dm=1,d=0.95,\theta=0.00001}$	26.80 %	3.46 %	10.21 %	3.41 %	10.00 %	3.36 %
$B_{2,\alpha=0.1,dm=20,d=0.95,\theta=0.00001}$	48.08 %	26.57 %	27.78 %	22.03 %	27.31 %	21.64 %
$B_{3,\alpha=0.1,dm=20,d=0.95,\theta=0.00001}$	98.12 %	76.19 %	97.06 %	91.32 %	97.03 %	91.39 %
$B_{4,\alpha=0.1,dm=20,d=0.95,\theta=0.00001}$	98.60 %	74.72 %	96.76 %	90.64 %	96.72 %	90.68 %

Table 6.15.: *Evaluation results of Newbob trainings with different learning rates α, data multiplications dm (by copying data), newbob weight decays d and newbob thresholds θ.*

6.7. Optimized Recognizer

All preprocessing steps and features that were useful were combined to create a recognizer that should perform best.

All models were much better than everything that was tried before. The results of this experiment show that single-symbol recognition with 369 classes and usual touch devices and the mouse can be done with a TOP1 error rate of 18.56 %, a TOP3 error of 4.11 % and a MER error rate of 4.01 %. This was achieved by a MLP with a 167 : 500 : 500 : 369 topology.

It used an algorithm to connect strokes of which the ends were less than 10 px away, scaled each recording to a unit square and shifted this unit square to $(0, 0)$. After that, a linear resampling step was applied to the first 4 strokes to resample them to 20 points each. All other strokes were discarded.

The 167 features were

- the first 4 strokes with 20 points per stroke resulting in 160 features,
- the re-curvature for the first 4 strokes,
- the ink,
- the number of strokes and
- the aspect ratio

SLP was applied with 1000 epochs per layer, a learning rate of $\eta = 0.1$ and a momentum of $\alpha = 0.1$. After that, the complete model was trained again for 1000 epochs with standard mini-batch gradient descent.

After the models $B_{1,c}$ – $B_{4,c}$ were trained the first 1000 epochs, they were trained again for 1000 epochs with a learning rate of $\eta = 0.05$. Table 6.16 shows that this improved the classifiers again.

System $B'_{2,c}$ had an error rate of 14.91 % if only the first symbol and its equivalence class were accepted as correct.

System	Classification error					
	TOP1	change	TOP3	change	MER	change
$B_{1,c}$	20.96 %	−2.38 %	5.24 %	−1.56 %	5.13 %	−1.51 %
$B_{2,c}$	18.26 %	−3.25 %	4.07 %	−1.68 %	3.98 %	−1.69 %
$B_{3,c}$	18.19 %	−3.74 %	4.06 %	−1.68 %	3.99 %	−1.65 %
$B_{4,c}$	18.57 %	−5.31 %	4.25 %	−1.87 %	4.18 %	−1.86 %
$B'_{1,c}$	19.33 %	−1.63 %	4.78 %	−0.46 %	4.67 %	−0.46 %
$B'_{2,c}$	17.52 %	−0.74 %	4.04 %	−0.03 %	3.96 %	−0.02 %
$B'_{3,c}$	17.65 %	−0.54 %	4.07 %	0.01 %	4.00 %	0.01 %
$B'_{4,c}$	17.82 %	−0.75 %	4.26 %	0.01 %	4.20 %	0.02 %

Table 6.16.: *Error rates of the optimized recognizer systems. The systems $B'_{i,c}$ were trained another* 1000 *epochs with a learning rate of $\eta = 0.05$. The value of the column "change" of the systems $B'_{i,c}$ is relative to $B_{i,c}$.*

6.8. User Interviews

Four people who used the recognizer on the website write-math.com with a Samsung Galaxy Note 10.1 (a 10.1 inch tablet with a stylus), with a smartphone and with a PC and a mouse were asked for feedback about the input devices. They were asked which device they prefer, why they prefer it, and if they could imagine entering multi-symbol formulas on those devices.

User A preferred this tablet with the stylus over using the mouse or a smartphone. She uses this tablet often and knows how to use the stylus for various applications. While using the website, she noted that the recognition works better if the symbol is written in a larger size. After noting that, she entered all symbols in a bigger size. She could imagine using the tablet to enter multi-symbol formulas.

User B used this tablet before, but did not use the stylus for writing before. He could enter symbols without problems with the tablet, but preferred using the index finger for writing instead of the styles. The reason is that he is used to lay the heel of the hand on the surface on which he is writing, which did not work well with the tablet. He preferred using a Nexus 4 smartphone to enter symbols. He could not imagine entering complex formulas with a tablet or the computer.

User C never used a touch device before. She preferred the tablet, but she had problems with the stylus as the tablet did not allow to lay the heel of the hand on the surface. This is the reason why she used the tablet with her fingers. She could also imagine to use a recognition system with multi-symbol formulas on a tablet, but not by using the mouse.

Like user C, user D never used a touch device before. He liked the stylus, although it took him a few minutes to get used to not placing the heel of the hand on the tablet. He also tried to enter symbols with a trackball, but that didn't work for him. According to user D, it is not possible to draw straight lines with a trackball which makes it unusable for writing symbols. He could also imagine to write multi-symbol formulas with a tablet.

6.9. Evaluation Summary

Five different classification systems were evaluated: A GTW classifier (see page 47) and four MLPs. The GTW classifier performed much worse than all MLP systems. For this reason, it was only tested in one variant.

The four MLPs $B_i, i \in \{ 1, 2, 3, 4 \}$ had i hidden layers with 500 neurons per layer. The baseline systems used scaling into a unit square, shifting to $(0, 0)$, 160 features that were coordinates of the first 20 points of the first 4 strokes. The baseline systems were trained with mini-batch gradient descent for 1000 epochs with a learning rate of $\eta = 0.1$ and a momentum of $\alpha = 0.1$.

For many recordings, there are at least two possibilities which symbols humans would recognize without context. This is the reason why the TOP1 error is less meaningful for single-symbol recognition. Hence two other error measures were calculated for every experiment: The TOP3 error and the MER error. The MER error of the baseline system B_2 is 5.67 %.

The MLPs were tested with five preprocessing algorithms, one data multiplication algorithm, five features and five variants for training in 16 separate experiments.

The effects of the preprocessing algorithms were often similar for the systems B_2 and B_3. The system B_4 was very sensitive to changes. A possible reason is the bigger number of weights, the higher order of internally computed features and the fact that the system was still able to improve after 1000 epochs of training. The system B_4 could be the best system if it was trained long enough as table 6.11 on page 48 shows. This means all results of the system B_4 should be taken with caution and eventually be evaluated again with 10 000 training epochs. System B_1 on the other hand performs much worse than system B_2. The storage size and the comparably small amount of necessary computing power to train B_1 are the only reasons to use this system.

The optimized system $B_{2,c}'$ was the best evaluated system with a TOP1 error rate of 17.52 %, a TOP3 error rate of 4.04 %, and a MER error rate of 3.96 %. This was achieved by extending the system B_2 by one algorithm or feature at a time, evaluating the extended system and combining all changes into $B_{2,c}'$ which improved the recognition rate.

The most important change was SLP (see page 50). It improved the MER error by 0.99 %. Adding the re-curvature feature improved the systems MER error by 0.57 %, the ink feature improved it by 0.53 %, the stroke count feature by 0.29 % and the aspect ratio feature by 0.07 %. The stroke connect preprocessing algorithm improved the MER error by 0.33 %, but all other preprocessing steps had either no effect that was bigger than random weight initialization or even made the classifiers worse. Douglas-Peucker smoothing is a preprocessing step that was not mentioned before in the literature for on-line HWR, but its evaluation results were much worse than the baseline system for any simplification threshold $\varepsilon > 0.05$. The computational cost of cubic spline interpolation is higher than linear interpolation and the classification results are worse.

7. Conclusion

7.1. Summary

The aim of this bachelor's thesis was to build a recognition system that can recognize many mathematical symbols with low error rates as well as to evaluate which preprocessing steps and features help to improve the recognition rate.

All recognition systems were trained and evaluated with 166 898 recordings for 369 symbols. These recordings were collected by two crowdsourcing projects (Detexify and write-math.com) and created with various devices. While some recordings were created with standard touch devices such as tablets and smartphones, others were created with the mouse.

MLPs were used for the classification task. Four baseline systems with different numbers of hidden layers were used, as the number of hidden layer influences the capabilities and problems of MLPs. Furthermore, an error measure MER was defined, which takes the top three hypotheses of the classifier, merges symbols such as `\sum` ($\sum$) and `\Sigma` (Σ) to equivalence classes, and then calculates the error.

All baseline systems used the same preprocessing queue. The recordings were scaled to fit into a unit square, shifted to $(0, 0)$, resampled with linear interpolation so that every stroke had exactly 20 points which are spread equidistant in time. The 80 (x, y) coordinates of the first 4 strokes were used to get exactly 160 input features for every recording. The baseline systems B_2 has a MER error of 5.67 %.

Three variations of the scale and shift algorithm, wild point filtering, stroke connect, weighted average smoothing, and Douglas-Peucker smoothing were evaluated. The evaluation showed that the scale and shift algorithm is extremely important and the connect strokes algorithm improves the classification. All other preprocessing algorithms either diminished the classification performance or had less influence on it than the random initialization of the MLPs weights.

Adding two slightly rotated variants for each recording and hence tripling the training set made the systems B_3 and B_4 perform much worse, but improved the performance of the smaller systems.

The global features re-curvature, ink, stoke count and aspect ratio improved the systems B_1–B_3, whereas the stroke center point feature made B_2 perform worse.

The learning rate and the momentum were evaluated. A learning rate of $\eta = 0.1$ and a momentum of $\alpha = 0.9$ gave the best results. Newbob training lead to much worse recognition rates. Denoising auto-encoders were evaluated as one way to use pretraining, but by this the error rate increased notably. However, supervised layer-wise pretraining improved the performance decidedly.

The stroke connect algorithm was added to the preprocessing steps of the baseline system as well as the re-curvature feature, the ink feature, the number of strokes and the aspect ratio. The training setup of the baseline system was changed to supervised layer-wise pretraining and the resulting model was trained with a lower learning rate again. This optimized recognizer $B'_{2,c}$ had a MER error of 3.96 %. This means that the MER error dropped by over 30 % in comparison to the baseline system B_2.

A MER error of 3.96 % makes the system usable for symbol lookup. It could also be used as a starting point for the development of a multiple-symbol classifier.

The aim of this bachelor's thesis was to develop a symbol recognition system which is easy to use, fast and has high recognition rates as well as evaluating ideas for single symbol classifiers. Some of those goals were reached. The recognition system $B'_{2,c}$ evaluates new recordings in a fraction of a second and has acceptable recognition rates. Many variations algorithms were evaluated. However, there are still many more algorithms which could be evaluated and, at the time of this work, the best classifier $B'_{2,c}$ is not publicly available.

7.2. Future Work

The presented system for single-symbol recognition with 369 classes works well. However, there are still many other symbols that one might want to classify, but which did not have enough training examples. This means that one part of the future work will include collecting more training examples, so that each class has at least 150 training examples.

Single-symbol recognition does already help LaTeX users a lot, but multiple symbol recognition is much more interesting. New hardware like iSketchnote or Equil Smartpen 2 can be used to improve the user experience of data input. It would be a significant development if users could employ those improved input devices to write complete formulas or systems of equations which a recognition system would record, recognize, and optimize for the best typeset result.

User interviews and surveys should be made to see how users employ such recognition systems, what they expect and if the system is useful for them. It might especially be interesting to see which kind of input device is comfortable for users and how the recorded data and the classification error changes with different devices.

However, there is still a lot that could be tried for single-symbol recognition with the mouse as an input device. Local features like bitmap environments notably improved recognition rates in earlier work and should be evaluated again and compared with the optimized recognizer B'_2. Different user interfaces like the one shown in figure 1.1 could be applied. Bottlenecks could be added to the MLP architecture. The F_1 score of preprocessing steps could be calculated to learn optimal parameters for noise reduction.

Future work could also attempt to find recognition systems that have less weights and similar recognition capabilities by inserting bottlenecks. If the neural net becomes smaller, it could be possible to let users download it in a JavaScript browser application and execute the classification directly on the client with ConvNetJS.

Bibliography

[Ara83] H. Arakawa, "On-line recognition of handwritten characters – Alphanumerics, Hiragana, Katakana, Kanji," *Pattern Recognition*, vol. 16, no. 1, pp. 9 – 21, 1983. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0031320383900031

[B$^+$06] C. M. Bishop *et al.*, *Pattern Recognition and Machine Learning*, M. Jordan, Ed. Springer Science+Business Media, 2006, vol. 1.

[BH84] A. Belaid and J.-P. Haton, "A syntactic approach for handwritten mathematical formula recognition," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. PAMI-6, no. 1, pp. 105–111, Jan 1984.

[BN72] P. W. Becker and K. A. Nielsen, "Pattern recognition using dynamic pictorial information," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. SMC-2, no. 3, pp. 434–437, July 1972. [Online]. Available: http://ieeexplore.ieee.org/xpl/abstractKeywords.jsp?arnumber=4309141

[Bro64] R. M. Brown, "On-line computer recognition of handprinted characters," *Electronic Computers, IEEE Transactions on*, vol. EC-13, no. 6, pp. 750–752, Dec 1964. [Online]. Available: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=4038313

[BS89] R. Bozinovic and S. Srihari, "Off-line cursive script word recognition," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 11, no. 1, pp. 68–83, Jan 1989. [Online]. Available: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=23114

[CRU$^+$08] N. A. Campbell, J. B. Reece, L. A. Urry, M. L. Cain, S. A. Wasserman, P. V. Minorsky, and R. B. Jackson, *Biology*, 8th ed., B. Wilbur, Ed. Pearson, 2008.

[Deea] "Denoising autoencoders (da)." [Online]. Available: http://deeplearning.net/tutorial/dA.html

[deeb] "Going from logistic regression to mlp." [Online]. Available: http://www.deeplearning.net/tutorial/mlp.html#going-from-logistic-regression-to-mlp

[DHS01] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification.* John Wiley & Sons, 2001.

[Dim58] T. L. Dimond, "Devices for reading handwritten characters," in *Papers and Discussions Presented at the December 9-13, 1957, Eastern Joint Computer Conference: Computers with Deadlines to Meet*, ser. IRE-ACM-AIEE '57 (Eastern). New York, NY, USA: ACM, 1958, pp. 232–237. [Online]. Available: http://doi.acm.org/10.1145/1457720.1457765

[GAC$^+$91] I. Guyon, P. Albrecht, Y. L. Cun, J. Denker, and W. Hubbard, "Design of a neural network character recognizer for a touch terminal," *Pattern Recognition*, vol. 24, no. 2, pp. 105 – 119, 1991. [Online]. Available: http://www.sciencedirect.com/science/article/pii/003132039190081F

[GP93] W. Guerfali and R. Plamondon, "Normalizing and restoring on-line handwriting," *Pattern Recognition*, vol. 26, no. 3, pp. 419–431, 1993. [Online]. Available: http://dx.doi.org/10.1016/0031-3203(93)90169-W

[Gro66] G. F. Groner, "Real-time recognition of handprinted text," in *Proceedings of the November 7-10, 1966, Fall Joint Computer Conference*, ser. AFIPS '66 (Fall). New York, NY, USA: ACM, 1966, pp. 591–601. [Online]. Available: http://doi.acm.org/10.1145/1464291.1464355

[HBT94] J. Hu, M. K. Brown, and W. Turin, "Handwriting recognition with hidden Markov models and grammatical constraints," in *In Proceedings of the Fourth International Workshop on Frontiers in Handwriting Recognition*, 1994. [Online]. Available: http://www.bell-labs.com/user/jianhu/papers/iwfhr94.ps

[Hin89] G. E. Hinton, "Connectionist learning procedures," *Artif. Intell.*, vol. 40, no. 1-3, pp. 185–234, Sep. 1989. [Online]. Available: http://dx.doi.org/10.1016/0004-3702(89)90049-0

[HK06] B. Huang and M.-T. Kechadi, "An HMM-SNN method for online handwriting symbol recognition," in *Image Analysis and Recognition*, ser. Lecture Notes in Computer Science, A. Campilho and M. Kamel, Eds. Springer Berlin Heidelberg, 2006, vol. 4142, pp. 897–905. [Online]. Available: http://dx.doi.org/10.1007/11867661_81

[HW89] I. Hampshire, J.B. and A. Waibel, "A novel objective function for improved phoneme recognition using time delay neural networks," in *Neural Networks, 1989. IJCNN., International Joint Conference on*, 1989, pp. 235–241 vol.1. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=118586

[HZK09] B. Q. Huang, Y. Zhang, and M.-T. Kechadi, "Preprocessing techniques for online handwriting recognition," in *Intelligent Text Categorization and Clustering*, ser. Studies in Computational Intelligence, N. Nedjah, L. de Macedo Mourelle, J. Kacprzyk, F. França, and A. de De Souza, Eds. Springer Berlin Heidelberg, 2009, vol. 164, ch. Preprocessing Techniques for Online Handwriting Recognition, pp. 25–45. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-85644-3_2

[iHY80] S. ichi Hanaki and T. Yamazaki, "On-line recognition of handprinted kanji characters," *Pattern Recognition*, vol. 12, no. 6, pp. 421 – 429, 1980. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0031320380900187

[IMP76] S. Impedovo, B. Marangelli, and V. L. Plantamura, "Real-time recognition of handwritten numerals," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. SMC-6, no. 2, pp. 145–148, Feb 1976. [Online]. Available: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5409186

[JMRW01] S. Jaeger, S. Manke, J. Reichert, and A. Waibel, "Online handwriting recognition: the NPen++ recognizer," in *International Journal on Document Analysis and Recognition*, 2001, pp. 169–180.

[JMW00] S. Jaeger, S. Manke, and A. Waibel, "NPen++: An on-line handwriting recognition system," in *in 7th International Workshop on Frontiers in Handwriting Recognition*, 2000, pp. 249–260. [Online]. Available: http://isl.anthropomatik.kit.edu/cmu-kit/IWFHR_stephen1.pdf

[KC98] A. Khotanzad and C. Chung, "Hand written digit recognition using combination of neural network classifiers," in *Image Analysis and Interpretation, 1998 IEEE Southwest Symposium on*, 4 1998, pp. 168–173. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=666880

[Kir10] D. Kirsch, "Detexify: Erkennung handgemalter LaTeX-symbole," Diploma thesis, Westfälische Wilhelms-Universität Münster, 10 2010. [Online]. Available: http://danielkirs.ch/thesis.pdf

[KR98] A. Kosmala and G. Rigoll, "Recognition of on-line handwritten formulas," in *In Proceedings of the Sixth International Workshop on Frontiers in Handwriting Recognition*, 1998, pp. 219–228. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.41.9056

[KRLP99] A. Kosmala, G. Rigoll, S. Lavirotte, and L. Pottier, "On-line handwritten formula recognition using hidden Markov models and context dependent graph grammars," in *Proceedings of the Fifth Internation Conference on Document Analysis and Recognition (ICDAR)*, 1999, pp. 107–110. [Online]. Available: http://hal.inria.fr/docs/00/56/46/45/PDF/kosmala-rigoll-etal_1999.pdf

[KWL95] M. Koschinski, H.-J. Winkler, and M. Lang, "Segmentation and recognition of symbols within handwritten mathematical expressions," in *Acoustics, Speech, and Signal Processing, 1995. ICASSP-95., 1995 International Conference on*, vol. 4, May 1995, pp. 2439–2442 vol.4.

[LBK$^+$08] H. Lodish, A. Berk, C. A. Kaiser, M. Krieger, M. P. Scott, A. Bretscher, H. Ploegh, and P. Matsudaira, *Molecular Cell Biology*, 6th ed., K. Ahr, Ed. W. H. Freeman and Company, 2008.

[MFW94] S. Manke, M. Finke, and A. Waibel, "Combining bitmaps with dynamic writing information for on-line handwriting recognition," in *Proceedings of the ICPR-94*, 1994, pp. 596–598.

[MFW95] ——, "The use of dynamic writing information in a connectionist on-line cursive handwriting recognition system," in *Advances in Neural Information Processing Systems 7*, G. Tesauro, D. Touretzky, and T. Leen, Eds. MIT Press, 1995, pp. 1093–1100. [Online]. Available: http://isl.anthropomatik.kit.edu/cmu-kit/downloads/The_Use_of_Dynamic_Writing_Information_in_a_Connectionist_On-Line_Cursive_Handwriting_Recognition_System(3).pdf

[Mit97] T. M. Mitchell, *Machine learning*, ser. McGraw Hill series in computer science. McGraw-Hill, 1997.

[MP43] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943. [Online]. Available: http://dx.doi.org/10.1007/BF02478259

[MVGK$^+$11] H. Mouchere, C. Viard-Gaudin, D. H. Kim, J. H. Kim, and U. Garain, "Crohme2011: Competition on recognition of online handwritten mathematical expressions," in *International Conference on Document*

Analysis and Recognition (ICDAR), 2011, Sept 2011, pp. 1497–1500. [Online]. Available: http://hal.archives-ouvertes.fr/docs/00/61/52/16/PDF/CROHME_CRC511.pdf

[MVGK$^+$12] H. Mouchere, C. Viard-Gaudin, D. Kim, J. Kim, and U. Garain, "Icfhr 2012 competition on recognition of on-line mathematical expressions (crohme 2012)," in *International Conference on Frontiers in Handwriting Recognition (ICFHR), 2012*, Sept 2012, pp. 811–816. [Online]. Available: http://hal.archives-ouvertes.fr/docs/00/71/78/50/PDF/Mouchere2012_CROHME.pdf

[MVGZ$^+$13] H. Mouchere, C. Viard-Gaudin, R. Zanibbi, U. Garain, D. H. Kim, and J. H. Kim, "Icdar 2013 crohme: Third international competition on recognition of online handwritten mathematical expressions," in *12th International Conference on Document Analysis and Recognition (ICDAR), 2013*, Aug 2013, pp. 1428–1432. [Online]. Available: http://www.isical.ac.in/~crohme/CROHME2013.pdf

[new00] "The training performed by qnstrn," 08 2000. [Online]. Available: http://www1.icsi.berkeley.edu/Speech/faq/nn-train.html

[Ng04] A. Y. Ng, "Feature selection, l1 vs. l2 regularization, and rotational invariance," in *Proceedings of the Twenty-first International Conference on Machine Learning*, ser. ICML '04. New York, NY, USA: ACM, 2004, pp. 78–. [Online]. Available: http://doi.acm.org/10.1145/1015330.1015435

[Pow73] V. M. Powers, "Pen direction sequences in character recognition," *Pattern Recognition*, vol. 5, no. 4, pp. 291 – 302, 1973. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0031320373900228

[Ros58] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-70911-1_20

[SBV96] B. Schölkopf, C. Burges, and V. Vapnik, "Incorporating invariances in support vector learning machines," in *Artificial Neural Networks – ICANN 96*, ser. Lecture Notes in Computer Science, C. von der Malsburg, W. von Seelen, J. Vorbrüggen, and B. Sendhoff, Eds., vol. 1112. Springer Berlin Heidelberg, 1996, pp. 47–52. [Online]. Available: http://link.springer.com/chapter/10.1007/3-540-61510-5_12

[SGH94] M. Schenkely, I. Guyonz, and D. Hendersonz, "On-line cursive script recognition using time delay neural networks and hidden Markov models," in *1994 IEEE International Conference on Acoustics, Speech, and Signal Processing.*, vol. ii, 4 1994, pp. 637–640. [Online]. Available: http://pdf.aminer.org/003/076/160/on_line_cursive_script_recognition_using_time_delay_neural_networks.pdf

[Tap87] C. C. Tappert, *Speed, Accuracy, Flexibility Trade-offs in On-line Character Recognition*, ser. Research report. IBM T.J. Watson Research Center, 1987. [Online]. Available: http://books.google.com/books?id=5br_HAAACAAJ

[TSW90] C. C. Tappert, C. Y. Suen, and T. Wakahara, "The state of the art in online handwriting recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 12, no. 8, pp. 787–808, 8 1990. [Online]. Available: http://dx.doi.org/10.1109/34.57669

[VLBM08] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, “Extracting and composing robust features with denoising autoencoders,” in *Proceedings of the 25th International Conference on Machine Learning*, ser. ICML ’08. New York, NY, USA: ACM, 2008, pp. 1096–1103. [Online]. Available: http://doi.acm.org/10.1145/1390156.1390294

[VW90] M. Visvalingam and J. D. Whyatt, “The douglas-peucker algorithm for line simplification: Re-evaluation through visualization,” in *Computer Graphics Forum*, vol. 9, no. 3. Wiley Online Library, 1990, pp. 213–225. [Online]. Available: http://www.bowdoin.edu/~ltoma/teaching/cs350/spring06/Lecture-Handouts/hershberger92speeding.pdf

Glossary

ANN artificial neural network. 19

CE cross entropy. 27, 28

CFM classification figure of merit. 27

CROHME Competition on Recognition of Online Handwritten Mathematical Expressions. 6

CUDA Compute Unified Device Architecture. 5

Detexify A system used for on-line handwritten symbol recognition which is described in [Kir10]. 3

DTW dynamic time warping. 6, 19, 22

GMM Gaussian mixture model. 21

GPU graphics processing unit. 5, 26

GTW greedy time warping. 21, 22, 54

HMM hidden Markov model. 6, 19

HWR handwriting recognition. 1, 2, 7, 21, 55

HWRT handwriting recognition toolkit. 34

hypothesis The recognition results which a classifier returns is called a hypothesis. In other words, it is the “guess” of a classifier. 37, 57

JSON JSON, short for JavaScript Object Notation, is a language-independent data format that can be used to transmit data between a server and a client in web applications. 7

LDA linear discriminant analysis. 3, 21, 24

line Geometric object that is infinitely long and defined by two points.. 15

line segment Geometric object that has finite length and defined by two points.. 19

MER An error measure which combines symbols to equivalence classes. It was introduced on page 37. 37

MLP multilayer perceptron. 21, 24–27, 29, 37, 47–49, 53, 54, 57, 58

MSE mean squared error. 27, 50

PCA principal component analysis. 3, 21, 24

PyPI Python Package Index. 32

reference Labeled data is used to evaluate classifiers. Those labels are called references. 37

SLP supervised layer-wise pretraining. 29, 50, 51, 53, 55

stroke The path the pen took from the point where the pen was put down to the point where the pen was lifted first. 7

SVM support vector machine. 21

symbol An atomic semantic entity. A more detailed description can be found in section 1.1. 1

TDNN time delay neural network. 5

YAML YAML is a human-readable data format that can be used for configuration files. 84, 85

Appendix

A. Algorithms

The following pseudo-code makes use of 0-indexed lists and arrays. The notation *abc*$[-1]$ means that the last element of the list or array *abc* is accessed. The notation *abc*$[2:5]$ is called a *slice* in Python. It creates a new list from the list *abc* that contains the elements with index 2, 3 and 4. The slice *abc*$[1:-1]$ means that a new list is created that contains all elements except for the first of the list *abc*.

Algorithm 2 Resampling

```
function RESAMPLING($pointlist$, $points\_per\_stroke$)
    $new\_pointlist \leftarrow$ LIST( )
    for all $stroke$ in $pointlist$ do
        $new\_stroke \leftarrow$ LIST( )
        if COUNT(stroke) $< 4$ then
            ▷ Don't do anything if there are less than 4 points
            $new\_stroke \leftarrow stroke$
        else
            $x, y, t \leftarrow$ LIST( ), LIST( ), LIST( )
            for all $point$ in $stroke$ do
                $x$.APPEND($point$['$x$'])
                $y$.APPEND($point$['$y$'])
                $t$.APPEND($point$['$time$'])
            $f_x \leftarrow$ INTERPOLATE($x, t$)          ▷ This could be linear interpolation,
            $f_y \leftarrow$ INTERPOLATE($y, t$)     ▷ cubic spline interpolation or something else
            $times \leftarrow$ SPACE_LINEAR($t[0], t[-1]$)
            for all $t$ in $times$ do
                $point \leftarrow$ POINT($f_x(t), f_y(t), t$)
                $new\_stroke$.APPEND($point$)
        $new\_pointlist$.APPEND($new\_stroke$)
    return $new\_pointlist$
```

Algorithm 3 Scale and shift a list of strokes to the $[-0.5, 0.5] \times [-0.5, 0.5]$ unit square

```
function SCALE_AND_SHIFT($pointlist$)
    ▷ Calculate bounding box
    $min_x, min_y \leftarrow pointlist[0][0]['x'], pointlist[0][0]['y']$
    $max_x, max_y \leftarrow pointlist[0][0]['x'], pointlist[0][0]['y']$
    $min_t \leftarrow pointlist[0][0]['t'], pointlist[0][0]['t']$
    for all $stroke$ in $pointlist$ do
        for all $p$ in $stroke$ do
            $min_x \leftarrow \textsc{min}(min_x, p['x'])$
            $max_x \leftarrow \textsc{max}(max_x, p['x'])$
            $min_y \leftarrow \textsc{min}(min_y, p['y'])$
            $max_y \leftarrow \textsc{max}(max_y, p['y'])$
            $min_t \leftarrow \textsc{min}(min_t, p['t'])$
    ▷ Calculate parameters for scaling and shifting to $[-0.5, 0.5] \times [-0.5, 0.5]$
    $width, height \leftarrow max_x - min_x + 1, max_y - min_y + 1$
    $factor_x, factor_y \leftarrow \frac{1}{width}, \frac{1}{height}$
    $factor \leftarrow \min(factor_x, factor_y)$
    $add_x, add_y \leftarrow \frac{width \cdot factor}{2}, \frac{height \cdot factor}{2}$
    ▷ Move every single point of a recording
    for all $stroke$ in $pointlist$ do
        for all $p$ in $stroke$ do
            $p['x'] \leftarrow (p['x'] - min_x) \cdot factor - add_x$
            $p['y'] \leftarrow (p['y'] - min_y) \cdot factor - add_y$
            $p['t'] \leftarrow p['t'] - min_t$
    return $pointlist$
```

Algorithm 4 Dot reduction

```
function DOT_REDUCTION($pointlist$, $\theta \in \mathbb{R}_{\geq 0}$)
    $new\_pointlist \leftarrow$ List( )
    for all $stroke$ in $pointlist$ do
        $new\_stroke \leftarrow stroke$
        ▷ Calculate maximum distance of two points in a stroke
        $max\_distance \leftarrow$ GET_MAX_DISTANCE($stroke$)
        ▷ Merge points of a stroke if the distance between them is below a threshold $\theta$
        if $max\_distance < \theta$ then
            $p \leftarrow$ GET_AVERAGE_POINT($stroke$)
            $new\_stroke \leftarrow$ List($p$)
        $new\_pointlist$.APPEND($new\_stroke$)
    return $new\_pointlist$
```

Algorithm 5 Dehooking

```
function $\textsc{dehook\_stroke}(\mathit{stroke}, \theta \in \mathbb{R}_{\geq 0})$
    if $\textsc{count}(\mathit{stroke}) < 3$ then
        return $\mathit{stroke}$
    else
        $\mathit{new\_stroke} \leftarrow \mathit{stroke}[0 : \textsc{count}(\mathit{stroke}) - 1]$ ▷ Get everything but the last point
        $\mathit{stroke} \leftarrow \mathit{stroke}[\textsc{count}(\mathit{stroke}) - 3 :]$    ▷ get the last 3 points
        $p \leftarrow \mathit{stroke}[-1]$    ▷ last point
        if $\textsc{calculate\_angle}(\mathit{stroke}) < \theta$ then
            $\mathit{new\_stroke}.\textsc{append}(p)$
        else
            $\mathit{new\_stroke} \leftarrow \textsc{dehook\_stroke}(\mathit{new\_stroke}, \theta)$
    return $\mathit{new\_pointlist}$
```

Algorithm 6 Weighted average smoothing

```
function $\textsc{weighted\_average\_smoothing}(\mathit{pointlist}, \theta = [\frac{1}{6}, \frac{4}{6}, \frac{1}{6}])$
    $\theta \leftarrow \frac{1}{\textsc{sum}(\theta)} \cdot \theta$    ▷ Normalize parameters to a sum of 1
    $\mathit{new\_pointlist} \leftarrow \textsc{List}(\ )$
    for all $\mathit{stroke}$ in $\mathit{pointlist}$ do
        $\mathit{tmp} \leftarrow \textsc{List}(\mathit{stroke}[0])$
        $\mathit{new\_pointlist}.\textsc{append}(\mathit{tmp})$
        if $\textsc{count}(\mathit{stroke}) > 1$ then
            for $i \leftarrow 1; i < \textsc{count}(\mathit{stroke}) - 1; i \leftarrow i + 1$ do
                $p \leftarrow \theta_0 \cdot \mathit{stroke}[i-1] + \theta_1 \cdot \mathit{stroke}[i] + \theta_2 \cdot \mathit{stroke}[i+1]$
                $\mathit{new\_pointlist}[-1].\textsc{append}(p)$
            $\mathit{new\_pointlist}[-1].\textsc{append}(\mathit{stroke}[-1])$
    return $\mathit{new\_pointlist}$
```

Algorithm 7 The Douglas-Peucker algorithm for stroke simplification.

```
function $\textsc{douglas\_peucker}(\mathit{stroke}$ as list of points, $\varepsilon \in \mathbb{R}_{\geq 0})$
    $S \leftarrow \{\ \mathit{stroke}[0], \mathit{stroke}[-1]\ \}$
    ▷ Calculate point that is furthest away from line $(\mathit{stroke}[0], \mathit{stroke}[-1])$
    $\mathit{pi}_{\max} \leftarrow 0$    ▷ Index of the point with highest distance
    $d_{\max} \leftarrow 0$    ▷ Distance of the point with highest distance
    for $i \leftarrow 1$; $i < \textsc{count}(\mathit{stroke}) - 1$; $i \leftarrow i + 1$ do
        ▷ Distance of the point $i$ to the line
        ▷ defined by the first and the last point of stroke
        $d \leftarrow \textsc{distance}(\mathit{stroke}[0], \mathit{stroke}[-1], \mathit{stroke}[i])$
        if $d > d_{\max}$ then
            $d_{\max} \leftarrow d$
            $\mathit{pi}_{\max} \leftarrow i$
    ▷ Recursively apply the algorithm
    if $d_{\max} > \varepsilon$ then
        $S_1 \leftarrow \textsc{douglas\_peucker}(\mathit{stroke}[0 : \mathit{pi}_{\max}], \varepsilon)$
        $S_2 \leftarrow \textsc{douglas\_peucker}(\mathit{stroke}[\mathit{pi}_{\max} : -1], \varepsilon)$
        $S \leftarrow S \cup S_1 \cup S_2$
    return $S$
```

Algorithm 8 Greedy matching as described in [Kir10]

```
$a \leftarrow$ next from $A$
$b \leftarrow$ next from $B$
$d \leftarrow \delta(a, b)$
$a' \leftarrow$ next from $A$
$b' \leftarrow$ next from $B$
while points left in $A \wedge$ points left in $B$ do
    $l, m, r \leftarrow \delta(a', b), \delta(a', b'), \delta(a, b')$
    $\mu \leftarrow$ min $\{l, m, r\}$
    $d \leftarrow d + \mu$
    if $l = \mu$ then
        $a \leftarrow a'$
        $a' \leftarrow$ next from $A$
    else if $r = \mu$ then
        $b \leftarrow b'$
        $b' \leftarrow$ next from $B$
    else
        $a \leftarrow a'$
        $b \leftarrow b'$
        $a' \leftarrow$ next from $A$
        $b' \leftarrow$ next from $B$
if no points left in $A$ then
    for all points $p$ in $B$ do
        $d \leftarrow d + \delta(a', p)$
else if no points left in $B$ then
    for all points $p$ in $A$ do
        $d \leftarrow d + \delta(b', p)$
```

B. Tables

Dot reduction	
Minimum distance threshold	$[0, \text{maximum point distance})$
Wild point filter	
Maximum speed threshold	$(0, \frac{\text{maximum point distance}}{\text{recording time}})$
Dehook	
Maximum angle threshold	$(0, 360]$
Smooth	
Smoothing factor p_{i-1}	$[0, 1]$
Smoothing factor p_i	$[0, 1]$
Smoothing factor p_{i+1}	$[0, 1]$
Douglas Peucker	
Epsilon	$[0, \text{minimum point distance})$
Scale	
Size	$(0, \infty) \times (0, \infty)$
Shift	
Center	$\{ \text{true}, \text{false} \}$
Shift target	$\mathbb{R}^2$
Stroke connect	
Minimum distance threshold	$[0, \text{maximum point distance})$
Resample	
Type	$\{ \text{linear}, \text{cubic} \}$
Points per stroke	$1, 2, \dots$

Table B.1.: *Preprocessing algorithms, their parameters and value ranges of those parameters. All of those algorithms are explained in section 3.2.*

Base symbol		equivalent symbols	
LaTeX	Rendered	LaTeX	Rendered
`\sum`	$\sum$	`$\Sigma$`	Σ
`\prod`	$\prod$	`$\Pi$`	Π
		`$\sqcap$`	$\sqcap$
`\coprod`	$\coprod$	`$\amalg$`	$\amalg$
		`$\sqcup$`	$\sqcup$
`\perp`	$\perp$	`$\bot$`	$\bot$
`\models`	$\models$	`$\vDash$`	$\vDash$
`|`	$\vert$	`\mid`	$\mid$
`\Delta`	Δ	`$\triangle$`	$\triangle$
		`$\vartriangle$`	$\vartriangle$
`\\|`	$\Vert$	`$\parallel$`	$\parallel$
`\ohm`	Ω	`$\Omega$`	Ω
`\setminus`	$\setminus$	`$\backslash$`	$\backslash$
`\checked`	$\checkmark$	`$\checkmark$`	$\checkmark$
`\&`	$\&$	`$\with$`	$\&$
`\#`	$\#$	`$\sharp$`	$\sharp$
`\S`	$\S$	`$\mathsection$`	$\S$
`\nabla`	∇	`\triangledown`	$\triangledown$
`\lhd`	$\lhd$	`$\triangleleft$`	$\triangleleft$
		`$\vartriangleleft$`	$\vartriangleleft$
`\oiint`	$\oiint$	`$\varoiint$`	$\oiint$
`\mathbb{R}`	$\mathbb{R}$	`$\mathds{R}$`	$\mathbb{R}$
`\mathbb{Q}`	$\mathbb{Q}$	`\mathds{Q}`	$\mathbb{Q}$
`\mathbb{Z}`	$\mathbb{Z}$	`\mathds{Z}`	$\mathbb{Z}$
`\mathcal{A}`	$\mathcal{A}$	`\mathscr{A}`	$\mathscr{A}$
`\mathcal{D}`	$\mathcal{D}$	`\mathscr{D}`	$\mathscr{D}$
`\mathcal{N}`	$\mathcal{N}$	`\mathscr{N}`	$\mathscr{N}$
`\mathcal{R}`	$\mathcal{R}$	`\mathscr{R}`	$\mathscr{R}$
`\propto`	$\propto$	`$\varpropto$`	$\varpropto$

Table B.2.: *Symbols that cannot be distinguished in handwriting. Those symbols were used to define equivalence classes for an error measure MER which is introduced on page 37.*

LaTeX	Rendered	LaTeX	Rendered
`\alpha`	α	`\propto`	$\propto$
		`\ltimes`	$\ltimes$
`0`	0	`O`	O
		`o`	o
		`\circ`	$\circ$
		`\degree`	°
		`\fullmoon`	○
`\epsilon`	ϵ	`$\varepsilon$`	ε
		`$\in$`	$\in$
		`$\mathcal{E}$`	$\mathcal{E}$
`\Lambda`	Λ	`$\wedge$`	$\wedge$
`\emptyset`	$\emptyset$	`\O`	Ø
		`\o`	ø
		`$\diameter$`	⌀
		`$\varnothing$`	$\varnothing$
`\rightarrow`	$\rightarrow$	`$\longrightarrow$`	$\longrightarrow$
		`$\shortrightarrow$`	→
`\Rightarrow`	$\Rightarrow$	`$\Longrightarrow$`	$\Longrightarrow$
`\Leftrightarrow`	$\Leftrightarrow$	`$\Longleftrightarrow$`	$\Longleftrightarrow$
`\mapsto`	$\mapsto$	`\longmapsto`	$\longmapsto$
`\mathbb{1}`	𝟙	`\mathds{1}`	𝟙
`\mathscr{L}`	$\mathscr{L}$	`\mathcal{L}`	$\mathcal{L}$
`\mathbb{Z}`	$\mathbb{Z}$	`\mathcal{Z}`	$\mathcal{Z}$
`\geq`	$\geq$	`\geqslant`	$\geqslant$
		`\succeq`	$\succeq$
`\leq`	$\leq$	`\leqslant`	$\leqslant$
`\pi`	π	`\Pi`	Π
`\psi`	ψ	`\Psi`	Ψ
`\phi`	ϕ	`\Phi`	Φ
		`\emptyset`	$\emptyset$
`\rho`	ρ	`\varrho`	ϱ
`\theta`	θ	`\Theta`	Θ
`\odot`	$\odot$	`\astrosun`	☉
`\cdot`	$\cdot$	`\bullet`	$\bullet$
`x`	x	`\times`	$\times$
		`X`	X
		`\chi`	χ
		`\mathcal{X}`	$\mathcal{X}$
`\beta`	β	`\ss`	ß
`\male`	♂	`\mars`	♂
`\female`	♀	`\venus`	♀
`\bowtie`	$\bowtie$	`\Bowtie`	⋈
`\diamond`	$\diamond$	`\diamondsuit`	$\diamondsuit$
		`\lozenge`	$\lozenge$
`\dots`	$\dots$	`\dotsc`	$\dotsc$
`\mathcal{T}`	$\mathcal{T}$	`\tau`	τ

Table B.3.: *Symbols that are extremely difficult to distinguish in handwriting (1).*

LaTeX	Rendered	LaTeX	Rendered
\mathcal{A}	$\mathcal{A}$	A	A
\mathcal{D}	$\mathcal{D}$	D	D
\mathcal{N}	$\mathcal{N}$	N	N
\mathcal{R}	$\mathcal{R}$	R	R
\varepsilon	ε	\mathcal{E}	$\mathcal{E}$

Table B.4.: *Symbols that are extremely difficult to distinguish in handwriting (2).*

LaTeX	Rendered	LaTeX	Rendered
\dots	...	\textellipsis	...
-	-	\textendash	–
		\textemdash	—
		\--	-
		\---	–
		\----	—
_	_	\textunderscore	_
i	i	!‘	¡
		\textexclamdown	¡
@	@	$\MVAt$	@
|	|	\shortmid	∣
		\textpipe	|
		\textbar	|
\degree	°	\textdegree	°

Table B.5.: *LaTeX symbols that were not evaluated, but also have confusion problems*

Base symbol		\n variant		\not variant	
LaTeX	Rendered	LaTeX	Rendered	LaTeX	Rendered
`=`	$=$	`\neq`	$\neq$	`\not=`	$\not=$
`\cong`	$\cong$	`\ncong`	$\ncong$	`\not\cong`	$\not\cong$
`\equiv`	$\equiv$	-	-	`\not\equiv`	$\not\equiv$
`\in`	$\in$	`\notin`	$\notin$	`\not\in`	$\not\in$
`\vDash`	$\vDash$	`\nvDash`	$\nvDash$	`\not\vDash`	$\not\vDash$
`\mid`	$\mid$	`\nmid`	$\nmid$	`\not\mid`	$\not\mid$
`\exists`	$\exists$	`\nexists`	$\nexists$	`\not\exists`	$\not\exists$
`\subseteq`	$\subseteq$	`\nsubseteq`	$\nsubseteq$	`\not\subseteq`	$\not\subseteq$
`\rightarrow`	$\rightarrow$	`\nrightarrow`	$\nrightarrow$	`\not\rightarrow`	$\not\rightarrow$
`\Rightarrow`	$\Rightarrow$	`\nRightarrow`	$\nRightarrow$	`\not\Rightarrow`	$\not\Rightarrow$

Table B.6.: *\n and \not variants of symbols.*

Symbol	Mean	σ	Symbol	Mean	σ
`\blacksquare`	9.22	3.86	`\male`	3.31	0.55
`\blacktriangleright`	6.86	2.25	`\parr`	3.30	0.47
`\bullet`	6.63	4.05	`\mathfrak{X}`	3.30	0.36
`\boxtimes`	5.64	0.70	`\leftmoon`	3.29	0.52
`\circledast`	5.27	0.70	`\sun`	3.28	0.68
`\boxplus`	5.19	0.50	`\mathfrak{S}`	3.27	0.52
`\otimes`	4.89	0.52	`\mathds{P}`	3.25	0.46
`\circledR`	4.77	0.60	`\notin`	3.15	0.53
`\oplus`	4.62	0.47	`\mars`	3.13	0.40
`\star`	4.28	1.21	`\fullmoon`	3.05	0.25
`\circledcirc`	4.28	0.60	`\degree`	3.04	0.33
`\clubsuit`	4.20	1.83	`\mathds{1}`	3.01	0.67
`\mathbb{Q}`	4.14	0.41	`\cong`	2.96	0.29
`\oiint`	4.12	0.55	`\mathds{C}`	2.95	0.47
`\copyright`	4.07	0.45	`\female`	2.68	0.55
`\Bowtie`	3.92	0.46	`\venus`	2.64	0.32
`\mathds{Q}`	3.89	0.53	`\ohm`	2.49	0.50
`\mathfrak{M}`	3.84	0.89	`\celsius`	2.42	0.37
`\mathds{R}`	3.83	0.50	`\sqrt{}`	1.89	0.38
`\mathds{E}`	3.82	0.63	`\checked`	1.59	0.19
`\mathds{Z}`	3.80	0.58	`\cdot`	0.77	2.19
`\mathds{N}`	3.78	0.50	`\therefore`	0.46	1.04
`\mathfrak{A}`	3.51	0.54	`\because`	0.36	1.01
`\astrosun`	3.39	0.61	`\dotsc`	0.06	0.30

Table B.7.: *Mean and standard deviation σ of the ink feature of symbols that are not shown in figure C.1.*

Symbol	Mean	σ	Symbol	Mean	σ
`\sun`	9.46	2.18	`\mathfrak{M}`	2.17	1.72
`\mathds{E}`	4.78	1.56	`\astrosun`	2.17	0.58
`\mathds{1}`	3.40	0.85	`\celsius`	2.05	0.29
`\female`	3.35	1.88	`\copyright`	2.03	0.22
`\mathfrak{X}`	3.26	1.60	`\diameter`	2.01	0.29
`\male`	3.25	3.73	`\mathfrak{A}`	1.45	0.92
`\mathds{P}`	3.17	0.62	`\Bowtie`	1.18	0.60
`\mathds{Q}`	3.15	2.73	`\leftmoon`	1.15	0.36
`\mathds{R}`	3.15	0.77	`\ohm`	1.05	0.29
`\cong`	3.04	0.30	`\mathfrak{S}`	1.05	0.27
`\venus`	3.02	0.22	`\parr`	1.05	0.22
`\mathds{N}`	3.00	1.03	`\sqrt{}`	1.04	0.29
`\mars`	2.70	1.04	`\checked`	1.04	0.22
`\dotsc`	2.57	1.13	`\degree`	1.03	0.16
`\mathds{Z}`	2.19	0.78	`\fullmoon`	1.02	0.14

Table B.8.: *Mean and standard deviation σ of the stroke count feature of symbols that are not shown in figure 6.4.*

Symbol	Mean	σ	Symbol	Mean	σ
`-`	33.09	31.82	`\mathfrak{S}`	1.04	0.25
`\dots`	21.35	23.35	`\copyright`	1.04	0.19
`\dotsc`	20.15	27.43	`\celsius`	1.01	0.25
`\rightharpoonup`	5.06	2.28	`\diameter`	0.99	0.29
`\multimap`	4.43	1.90	`\mars`	0.99	0.20
`\longrightarrow`	4.33	2.27	`\mathfrak{A}`	0.97	0.33
`\frown`	3.65	1.71	`\mathds{Q}`	0.96	0.21
`\twoheadrightarrow`	3.60	1.31	`\mathfrak{X}`	0.93	0.26
`\rightsquigarrow`	3.44	1.15	`\male`	0.92	0.40
`\sim`	3.41	1.16	`\mathds{C}`	0.90	0.21
`\leadsto`	3.32	1.12	`\mathds{E}`	0.84	0.23
`\ohm`	1.61	0.53	`\parr`	0.75	0.15
`\cong`	1.52	0.41	`\mathds{1}`	0.72	0.24
`\sqrt{}`	1.50	0.56	`\mathds{N}`	0.72	0.18
`\mathfrak{M}`	1.28	0.35	`\female`	0.71	0.38
`\mathds{Z}`	1.18	0.29	`\mathds{R}`	0.71	0.17
`\Bowtie`	1.15	0.33	`\mathds{P}`	0.64	0.21
`\checked`	1.13	0.56	`\venus`	0.60	0.13

Table B.9.: *Mean and standard deviation of the aspect ratio of symbols that are not shown in figure C.2.*

B.1. Evaluated Symbols

LaTeX	Rendered	LaTeX	Rendered
\&	&	\nmid	∤
\Im	ℑ	\nvDash	⊭
\Re	ℜ	\int	∫
\S	§	\fint	⨏
\Vdash	⊩	\odot	⊙
\aleph	ℵ	\oiint	∯
\amalg	⨿	\oint	∮
\angle	∠	\varoiint	∯
\ast	∗	\ominus	⊖
\asymp	≍	\oplus	⊕
\backslash	\	\otimes	⊗
\between	≬	\parallel	∥
\blacksquare	■	\parr	⅋
\blacktriangleright	▶	\partial	∂
\bot	⊥	\perp	⊥
\bowtie	⋈	\pitchfork	⋔
\boxdot	⊡	\pm	±
\boxplus	⊞	\prime	′
\boxtimes	⊠	\prod	∏
\bullet	•	\propto	∝
\checkmark	✓	\rangle	⟩
\circ	∘	\rceil	⌉
\circledR	®	\rfloor	⌋
\circledast	⊛	\rrbracket	⟧
\circledcirc	⊚	\rtimes	⋊
\clubsuit	♣	\sharp	♯
\coprod	∐	\sphericalangle	∢
\copyright	©	\sqcap	⊓
\dag	†	\sqcup	⊔
\dashv	⊣	\sqrt{}	√
\diamond	⋄	\square	□
\diamondsuit	♢	\star	⋆
\div	÷	\sum	∑
\ell	ℓ	\times	×
\flat	♭	\top	⊤
\frown	⌢	\triangle	△
\guillemotleft	´	\triangledown	▽
\hbar	ℏ	\triangleleft	◁
\heartsuit	♡	\trianglelefteq	⊴
\infty	∞	\triangleq	≜
\langle	⟨	\triangleright	▷
\lceil	⌈	\uplus	⊎
\lfloor	⌊	\vDash	⊨
\lhd	◁	\varnothing	∅
\lightning	↯	\varpropto	∝
\llbracket	⟦	\vartriangle	△
\lozenge	◊	\vdash	⊢
\ltimes	⋉	\with	&

Continued on next page

LaTeX	Rendered	LaTeX	Rendered
\mathds{1}	$\mathds{1}$	\wp	$\wp$
\mathsection	$\S$	\wr	$\wr$
\mid	$\mid$	\\{	$\{$
\models	$\models$	\\\|	$\\|$
\mp	$\mp$	\\}	$\}$
\multimap	$\multimap$	\vee	$\vee$
\nabla	∇	\wedge	$\wedge$
\neg	$\neg$	\barwedge	$\barwedge$

Table B.10.: *112 symbols that were used for evaluation.*

LaTeX	Rendered	LaTeX	Rendered	LaTeX	Rendered	LaTeX	Rendered
\#	$\#$	A	A	S	S	i	i
\\$	$\$$	B	B	T	T	j	j
\%	$\%$	C	C	U	U	k	k
+	$+$	D	D	V	V	l	l
-	$-$	E	E	W	W	m	m
/	$/$	F	F	X	X	n	n
0	0	G	G	Y	Y	o	o
1	1	H	H	Z	Z	p	p
2	2	I	I	[	$[$	q	q
3	3	J	J	]	$]$	r	r
4	4	K	K	a	a	s	s
5	5	L	L	b	b	u	u
6	6	M	M	c	c	v	v
7	7	N	N	d	d	w	w
8	8	O	O	e	e	x	x
9	9	P	P	f	f	y	y
<	$<$	Q	Q	g	g	z	z
>	$>$	R	R	h	h	|	$\vert$

Table B.11.: *72 ASCII symbols that were used for evaluation, including all ten digits, the Latin alphabet in lower and upper case and a few more symbols.*

LaTeX	Rendered	LaTeX	Rendered	LaTeX	Rendered
\approx	$\approx$	\geqslant	$\geqslant$	\lesssim	$\lesssim$
\doteq	$\doteq$	\neq	$\neq$	\backsim	$\backsim$
\simeq	$\simeq$	\not\equiv	$\not\equiv$	\sim	$\sim$
\equiv	$\equiv$	\preccurlyeq	$\preccurlyeq$	\succ	$\succ$
\geq	$\geq$	\preceq	$\preceq$	\prec	$\prec$
\leq	$\leq$	\succeq	$\succeq$	\gtrless	$\gtrless$
\leqslant	$\leqslant$	\gtrsim	$\gtrsim$	\cong	$\cong$

Table B.12.: *21 symbols that were used for evaluation and indicate a relationship.*

LaTeX	Rendered	LaTeX	Rendered
\Downarrow	$\Downarrow$	\nrightarrow	$\nrightarrow$
\Leftarrow	$\Leftarrow$	\rightarrow	$\rightarrow$
\Leftrightarrow	$\Leftrightarrow$	\rightleftarrows	$\rightleftarrows$
\Longleftrightarrow	$\Longleftrightarrow$	\rightrightarrows	$\rightrightarrows$
\Longrightarrow	$\Longrightarrow$	\rightsquigarrow	$\rightsquigarrow$
\Rightarrow	$\Rightarrow$	\searrow	$\searrow$
\circlearrowleft	$\circlearrowleft$	\shortrightarrow	$\shortrightarrow$
\circlearrowright	$\circlearrowright$	\twoheadrightarrow	$\twoheadrightarrow$
\curvearrowright	$\curvearrowright$	\uparrow	$\uparrow$
\downarrow	$\downarrow$	\rightharpoonup	$\rightharpoonup$
\hookrightarrow	$\hookrightarrow$	\rightleftharpoons	$\rightleftharpoons$
\leftarrow	$\leftarrow$	\longmapsto	$\longmapsto$
\leftrightarrow	$\leftrightarrow$	\mapsfrom	$\mapsfrom$
\longrightarrow	$\longrightarrow$	\mapsto	$\mapsto$
\nRightarrow	$\nRightarrow$	\leadsto	$\leadsto$
\nearrow	$\nearrow$	\upharpoonright	$\upharpoonright$

Table B.13.: *32 arrow symbols that were used for evaluation.*

LaTeX	Rendered	LaTeX	Rendered	LaTeX	Rendered
\alpha	α	\xi	ξ	\Xi	Ξ
\beta	β	\pi	π	\Pi	Π
\gamma	γ	\rho	ρ	\Sigma	Σ
\delta	δ	\sigma	σ	\Phi	Φ
\epsilon	ϵ	\tau	τ	\Psi	Ψ
\zeta	ζ	\phi	ϕ	\Omega	Ω
\eta	η	\chi	χ	\varepsilon	ε
\theta	θ	\psi	ψ	\varkappa	$\varkappa$
\iota	ι	\omega	ω	\varpi	ϖ
\kappa	κ	\Gamma	Γ	\varrho	ϱ
\lambda	λ	\Delta	Δ	\varphi	φ
\mu	μ	\Theta	Θ	\vartheta	ϑ
\nu	ν	\Lambda	Λ		

Table B.14.: *All Greek letters and some variations of Greek letters were used for evaluation. 38 of them are in this table, the rest is identical to Latin letters.*

LaTeX	Rendered	LaTeX	Rendered	LaTeX	Rendered
\mathcal{A}	$\mathcal{A}$	\mathcal{T}	$\mathcal{T}$	\mathds{Z}	$\mathbb{Z}$
\mathcal{B}	$\mathcal{B}$	\mathcal{U}	$\mathcal{U}$	\mathfrak{A}	$\mathfrak{A}$
\mathcal{C}	$\mathcal{C}$	\mathcal{X}	$\mathcal{X}$	\mathfrak{M}	$\mathfrak{M}$
\mathcal{D}	$\mathcal{D}$	\mathcal{Z}	$\mathcal{Z}$	\mathfrak{S}	$\mathfrak{S}$
\mathcal{E}	$\mathcal{E}$	\mathbb{H}	$\mathbb{H}$	\mathfrak{X}	$\mathfrak{X}$
\mathcal{F}	$\mathcal{F}$	\mathbb{N}	$\mathbb{N}$	\mathscr{A}	$\mathscr{A}$
\mathcal{G}	$\mathcal{G}$	\mathbb{Q}	$\mathbb{Q}$	\mathscr{C}	$\mathscr{C}$
\mathcal{H}	$\mathcal{H}$	\mathbb{R}	$\mathbb{R}$	\mathscr{D}	$\mathscr{D}$
\mathcal{L}	$\mathcal{L}$	\mathbb{Z}	$\mathbb{Z}$	\mathscr{E}	$\mathscr{E}$
\mathcal{M}	$\mathcal{M}$	\mathds{C}	$\mathbb{C}$	\mathscr{F}	$\mathscr{F}$
\mathcal{N}	$\mathcal{N}$	\mathds{E}	$\mathbb{E}$	\mathscr{H}	$\mathscr{H}$
\mathcal{O}	$\mathcal{O}$	\mathds{N}	$\mathbb{N}$	\mathscr{L}	$\mathscr{L}$
\mathcal{P}	$\mathcal{P}$	\mathds{P}	$\mathbb{P}$	\mathscr{P}	$\mathscr{P}$
\mathcal{R}	$\mathcal{R}$	\mathds{Q}	$\mathbb{Q}$	\mathscr{S}	$\mathscr{S}$
\mathcal{S}	$\mathcal{S}$	\mathds{R}	$\mathbb{R}$		

Table B.15.: *44 variants of Latin letters were used for evaluation.*

LaTeX	Rendered	LaTeX	Rendered	LaTeX	Rendered
\therefore	$\therefore$	\cdot	$\cdot$	\dots	$\dots$
\because	$\because$	\vdots	$\vdots$	\ddots	$\ddots$
\dotsc	$\dotsc$				

Table B.16.: *7 symbols that contain only dots were used for evaluation.*

LaTeX	R	LaTeX	R	LaTeX	R	LaTeX	R	LaTeX	R
\AA	Å	\L	Ł	\male	♂	\ohm	Ω	\sun	☼
\AE	Æ	\O	Ł	\mars	♂	\fullmoon	○	\degree	°
\aa	å	\o	ø	\female	♀	\leftmoon	☾	\iddots	⋰
\ae	Æ	\Bowtie	⋈	\venus	♀	\checked	✓	\diameter	⌀
\ss	ß	\celsius	°C	\astrosun	☉	\pounds	£	\mathbb{1}	𝟙

Table B.17.: *25 symbols that were used for evaluation.*

LaTeX	Rendered	LaTeX	Rendered	LaTeX	Rendered
\cup	$\cup$	\varsubsetneq	$\varsubsetneq$	\exists	$\exists$
\cap	$\cap$	\nsubseteq	$\nsubseteq$	\nexists	$\nexists$
\emptyset	$\emptyset$	\sqsubseteq	$\sqsubseteq$	\forall	$\forall$
\setminus	$\setminus$	\subseteq	$\subseteq$	\in	$\in$
\supset	$\supset$	\subsetneq	$\subsetneq$	\ni	$\ni$
\subset	$\subset$	\supseteq	$\supseteq$	\notin	$\notin$

Table B.18.: *18 set related symbols that were used for evaluation.*

B.2. Evaluation Results

System	Classification error		
	std	TOP3	merged
B_1	23.34 %	6.80 %	6.64 %
B_1	23.12 %	6.71 %	6.58 %
B_1	23.44 %	6.72 %	6.57 %
B_1	23.18 %	6.67 %	6.54 %
B_1	23.08 %	6.75 %	6.64 %
B_2	21.51 %	5.75 %	5.67 %
B_2	21.45 %	5.68 %	5.60 %
B_2	21.80 %	5.74 %	5.66 %
B_2	21.83 %	5.75 %	5.68 %
B_2	21.58 %	5.75 %	5.66 %
B_3	21.93 %	5.74 %	5.64 %
B_3	22.28 %	5.82 %	5.75 %
B_3	21.80 %	5.74 %	5.58 %
B_3	21.74 %	5.50 %	5.41 %
B_3	21.54 %	5.50 %	5.41 %
B_4	23.88 %	6.12 %	6.04 %
B_4	24.84 %	6.44 %	6.21 %
B_4	23.84 %	6.17 %	6.02 %
B_4	23.93 %	6.31 %	6.13 %
B_4	23.19 %	5.98 %	5.83 %

Table B.19.: *The influence of random weight initialization. This table is summed up on page 38.*

System	Classification error					
	TOP1	change	TOP3	change	MER	change
B_1	23.34 %		6.80 %		6.64 %	
B_2	21.51 %		5.75 %		5.67 %	
B_3	21.93 %		5.74 %		5.64 %	
B_4	23.88 %		6.12 %		6.04 %	
$B_{1,NSS}$	40.51 %	17.17 %	17.83 %	11.03 %	17.16 %	10.52 %
$B_{2,NSS}$	32.27 %	10.76 %	10.19 %	4.44 %	10.07 %	4.40 %
$B_{3,NSS}$	34.31 %	12.38 %	11.66 %	5.92 %	11.52 %	5.88 %
$B_{4,NSS}$	60.33 %	36.45 %	32.34 %	26.22 %	31.34 %	25.30 %
$B_{1,I2}$	22.75 %	−0.59 %	6.40 %	−0.40 %	6.24 %	−0.40 %
$B_{2,I2}$	21.52 %	0.01 %	5.42 %	−0.33 %	5.31 %	−0.36 %
$B_{3,I2}$	21.73 %	−0.20 %	5.20 %	−0.54 %	5.09 %	−0.55 %
$B_{4,I2}$	28.20 %	4.32 %	7.77 %	1.65 %	7.29 %	1.25 %
$B_{1,I3}$	22.77 %	−0.57 %	6.67 %	−0.13 %	6.53 %	−0.11 %
$B_{2,I3}$	21.86 %	0.35 %	5.87 %	0.12 %	5.80 %	0.13 %
$B_{3,I3}$	21.80 %	−0.13 %	5.95 %	0.21 %	5.84 %	0.20 %
$B_{4,I3}$	23.55 %	−0.33 %	6.11 %	−0.01 %	6.03 %	−0.01 %

Table B.20.: *The baseline models B_1–B_4 were tested with all three implementations of the scale and shift preprocessing algorithm. After every error score is indicated how much the system changed in comparison to its baseline. A change of* −0.05 % *means that the system improved by* 0.05 % *compared to its baseline system. The column "change" was left blank as implementation 1 was used in the baseline systems. Implementation 2 does not center the recording and implementation 3 does center the recording on both axes. The NSS models used no scale and shift algorithm. The results of this table are discussed on page 38.*

System	Classification error					
	TOP1	change	TOP3	change	MER	change
$B_{1,\theta_{sc}=5\,\text{px}}$	23.27 %	−0.07 %	6.50 %	−0.30 %	6.37 %	−0.27 %
$B_{2,\theta_{sc}=5\,\text{px}}$	21.20 %	−0.31 %	5.59 %	−0.16 %	5.50 %	−0.17 %
$B_{3,\theta_{sc}=5\,\text{px}}$	21.80 %	−0.13 %	5.54 %	−0.20 %	5.47 %	−0.17 %
$B_{4,\theta_{sc}=5\,\text{px}}$	24.29 %	0.41 %	6.10 %	−0.02 %	5.94 %	−0.10 %
$B_{1,\theta_{sc}=10\,\text{px}}$	23.17 %	−0.17 %	6.61 %	−0.19 %	6.47 %	−0.17 %
$B_{2,\theta_{sc}=10\,\text{px}}$	20.97 %	−0.54 %	5.43 %	−0.32 %	5.34 %	−0.33 %
$B_{3,\theta_{sc}=10\,\text{px}}$	21.34 %	−0.59 %	5.42 %	−0.32 %	5.33 %	−0.31 %
$B_{4,\theta_{sc}=10\,\text{px}}$	23.50 %	−0.38 %	6.11 %	−0.01 %	5.81 %	−0.23 %
$B_{1,\theta_{sc}=20\,\text{px}}$	22.81 %	−0.53 %	6.28 %	−0.52 %	6.19 %	−0.45 %
$B_{2,\theta_{sc}=20\,\text{px}}$	21.61 %	0.10 %	5.79 %	0.04 %	5.69 %	0.02 %
$B_{3,\theta_{sc}=20\,\text{px}}$	21.71 %	−0.22 %	5.55 %	−0.19 %	5.45 %	−0.19 %
$B_{4,\theta_{sc}=20\,\text{px}}$	24.36 %	0.48 %	6.23 %	0.11 %	5.93 %	−0.11 %

Table B.21.: *The baseline models B_1–B_4 with additionally applied stroke connect algorithm, before the scale and shift algorithm with different thresholds θ_{sc}. The results of this table are discussed on page 40.*

System	Classification error					
	TOP1	change	TOP3	change	MER	change
$B_{1,\varepsilon=0.05,\text{linear}}$	22.87 %	−0.47 %	6.68 %	−0.12 %	6.55 %	−0.09 %
$B_{2,\varepsilon=0.05,\text{linear}}$	21.24 %	−0.27 %	5.67 %	−0.08 %	5.57 %	−0.10 %
$B_{3,\varepsilon=0.05,\text{linear}}$	21.88 %	−0.05 %	6.01 %	0.27 %	5.92 %	0.28 %
$B_{4,\varepsilon=0.05,\text{linear}}$	23.84 %	−0.04 %	6.58 %	0.46 %	6.25 %	0.21 %
$B_{1,\varepsilon=0.05,\text{cubic}}$	25.26 %	1.92 %	8.77 %	1.97 %	8.73 %	2.09 %
$B_{2,\varepsilon=0.05,\text{cubic}}$	23.84 %	1.91 %	7.59 %	1.84 %	7.54 %	1.87 %
$B_{3,\varepsilon=0.05,\text{cubic}}$	23.95 %	2.02 %	7.49 %	1.75 %	7.42 %	1.78 %
$B_{4,\varepsilon=0.05,\text{cubic}}$	29.47 %	6.13 %	9.96 %	3.84 %	9.68 %	3.64 %
$B_{1,\varepsilon=0.1,\text{linear}}$	23.81 %	0.47 %	7.04 %	0.24 %	6.91 %	0.27 %
$B_{2,\varepsilon=0.1,\text{linear}}$	22.02 %	0.51 %	5.95 %	0.20 %	5.88 %	0.21 %
$B_{3,\varepsilon=0.1,\text{linear}}$	22.10 %	0.17 %	5.80 %	0.06 %	5.72 %	0.08 %
$B_{4,\varepsilon=0.1,\text{linear}}$	25.05 %	1.17 %	6.78 %	0.66 %	6.42 %	0.38 %
$B_{1,\varepsilon=0.2,\text{linear}}$	28.08 %	4.74 %	8.60 %	1.80 %	8.48 %	1.84 %
$B_{2,\varepsilon=0.2,\text{linear}}$	25.82 %	4.31 %	7.38 %	1.63 %	7.24 %	1.57 %
$B_{3,\varepsilon=0.2,\text{linear}}$	26.72 %	4.79 %	7.42 %	1.68 %	7.27 %	1.63 %
$B_{4,\varepsilon=0.2,\text{linear}}$	28.36 %	4.48 %	7.90 %	1.78 %	7.76 %	1.72 %
$B_{1,\varepsilon=0.2,\text{cubic}}$	30.98 %	7.64 %	10.77 %	3.97 %	10.62 %	3.98 %
$B_{2,\varepsilon=0.2,\text{cubic}}$	28.54 %	7.03 %	9.16 %	3.41 %	9.06 %	3.39 %
$B_{3,\varepsilon=0.2,\text{cubic}}$	28.94 %	7.01 %	8.82 %	3.08 %	8.66 %	3.02 %
$B_{4,\varepsilon=0.2,\text{cubic}}$	32.80 %	8.92 %	10.25 %	4.13 %	9.85 %	3.81 %

Table B.22.: *The evaluation results of Douglas-Peucker smoothing show that a strong simplification (a high ε value) gives much worse results. Cubic spline interpolation performed much worse than linear interpolation. Those results are explained on page 42.*

System	Classification error					
	TOP1	change	TOP3	change	MER	change
$B_{1,\eta=0.05}$	24.58 %	1.24 %	7.95 %	1.15 %	7.70 %	1.06 %
$B_{1,\eta=0.1}$	23.34 %		6.80 %		6.64 %	
$B_{1,\eta=0.2}$	23.41 %	0.07 %	6.66 %	−0.14 %	6.64 %	0.00 %
$B_{1,\eta=1}$	30.80 %	7.46 %	12.09 %	5.29 %	11.36 %	4.72 %
$B_{2,\eta=0.05}$	22.37 %	0.86 %	6.24 %	0.49 %	6.14 %	0.47 %
$B_{2,\eta=0.1}$	21.51 %		5.75 %		5.67 %	
$B_{2,\eta=0.2}$	22.39 %	0.88 %	6.02 %	0.27 %	5.95 %	0.28 %
$B_{2,\eta=1}$	30.27 %	8.76 %	12.80 %	7.05 %	11.29 %	5.62 %
$B_{3,\eta=0.05}$	22.81 %	0.88 %	6.01 %	0.27 %	5.89 %	0.25 %
$B_{3,\eta=0.1}$	21.93 %		5.74 %		5.64 %	
$B_{3,\eta=0.2}$	21.77 %	−0.16 %	5.83 %	0.09 %	5.71 %	0.07 %
$B_{3,\eta=1}$	90.67 %	68.74 %	86.98 %	81.24 %	86.12 %	80.48 %
$B_{4,\eta=0.05}$	25.23 %	1.94 %	6.86 %	0.74 %	6.74 %	0.70 %
$B_{4,\eta=0.1}$	23.88 %		6.12 %		6.04 %	
$B_{4,\eta=0.2}$	23.29 %	−0.59 %	6.14 %	0.02 %	5.98 %	−0.06 %
$B_{4,\eta=1}$	99.17 %	75.29 %	98.85 %	92.73 %	98.85 %	92.81 %

Table B.23.: *Evaluation results of the systems B_1 – B_4 with adjusted learning rates η. The column “change” was left blank for the baseline systems ($\eta = 0.1$), as this value will only be different from exactly 0 due to random weight initialization. The results of this table are explained on page 48.*

C. Figures

C.1. Scatterplots of Features

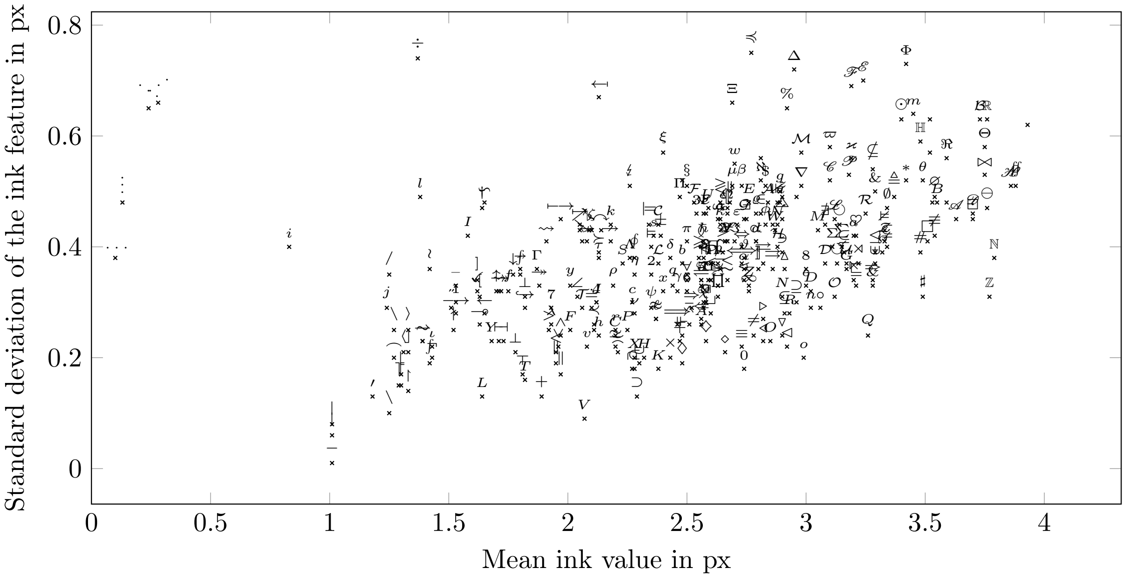


Figure C.1.: *Mean-standard deviation scatterplot of the ink feature. Some symbols were excluded from this plot. They are listed in table B.7. This type of scatterplot was introduced on page 43.*

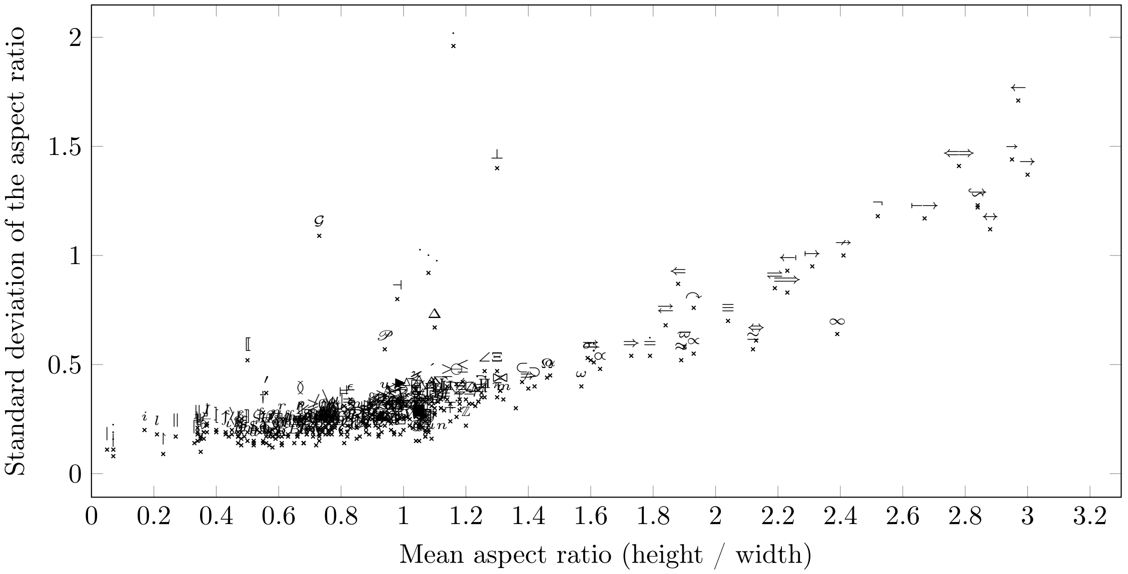
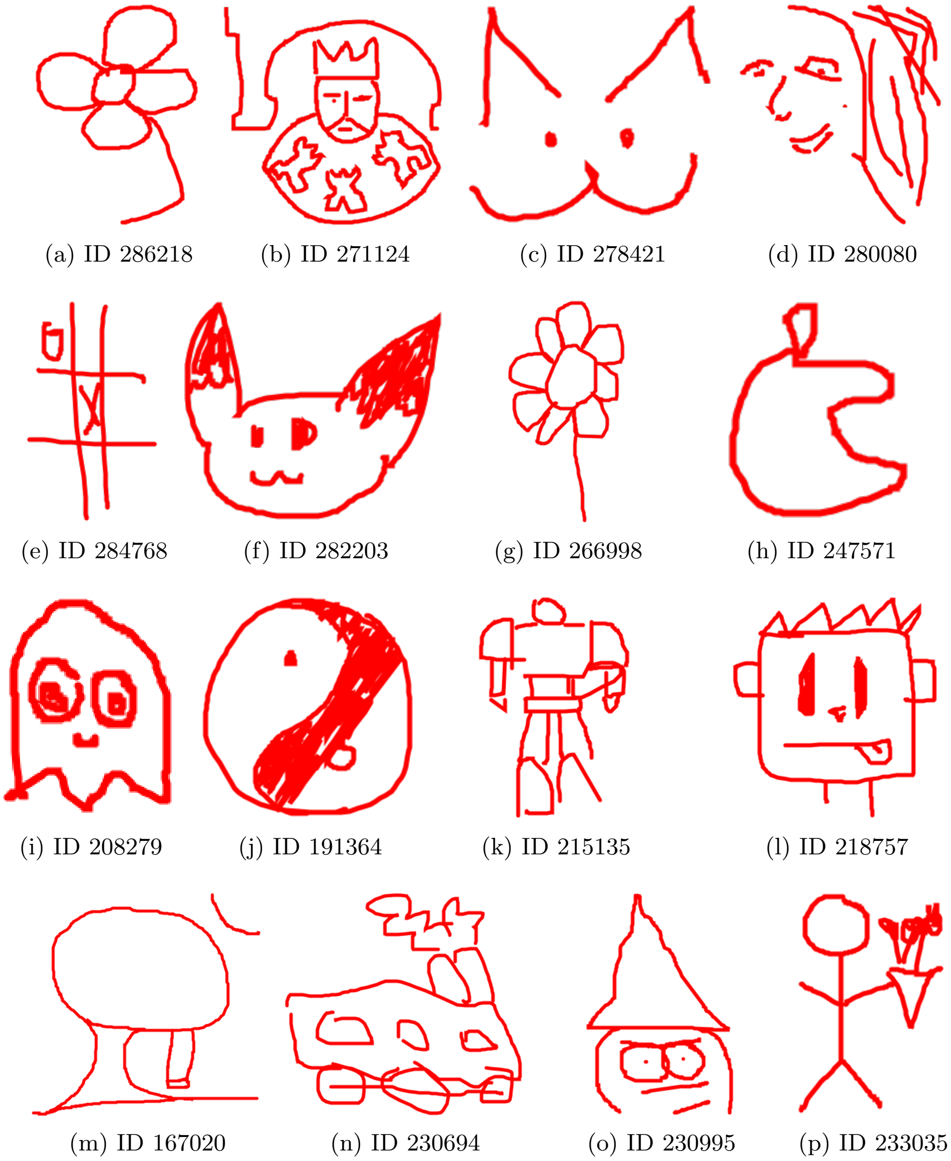


Figure C.2.: *Mean-standard deviation scatterplot of the aspect ratio feature. This type of scatterplot was introduced on page 43.*

D. Creative Users

The following drawings made some creative users:



(a) ID 286218 (b) ID 271124 (c) ID 278421 (d) ID 280080

(e) ID 284768 (f) ID 282203 (g) ID 266998 (h) ID 247571

(i) ID 208279 (j) ID 191364 (k) ID 215135 (l) ID 218757

(m) ID 167020 (n) ID 230694 (o) ID 230995 (p) ID 233035

Figure D.3.: *Images drawn by creative users.*

E. Raw Data Example

The following code shows the recording with ID 292927 as it is stored in the database. It is a JSON string that contains a list of strokes. Every stroke is a list of control points where every control point has the `x` and `y` coordinates as well as the `time`.

The symbol that was drawn is a $\subseteq$. So it has two strokes. This recording has 145 control points.

The Unix time of the 28th of April 2014, 3 p.m. UTC would be 1 398 636 000. The Unix time of 1 411 732 873 010 is the number of milliseconds since 1970. It is the 26th of September 2014 at 12:01:13 p.m. UTC.

292927.json

```
[[{"x":657,"y":600,"time":1411732873010},
  {"x":656,"y":600,"time":1411732873056},
  {"x":654,"y":599,"time":1411732873064},
  {"x":651,"y":599,"time":1411732873072},
  {"x":650,"y":598,"time":1411732873078},
  {"x":646,"y":598,"time":1411732873086},
  {"x":643,"y":598,"time":1411732873094},
  {"x":638,"y":598,"time":1411732873102},
  {"x":634,"y":598,"time":1411732873110},
  {"x":629,"y":598,"time":1411732873118},
  {"x":626,"y":597,"time":1411732873126},
  {"x":620,"y":597,"time":1411732873134},
  {"x":616,"y":597,"time":1411732873142},
  {"x":614,"y":597,"time":1411732873150},
  {"x":612,"y":596,"time":1411732873158},
  {"x":606,"y":596,"time":1411732873164},
  {"x":603,"y":596,"time":1411732873172},
  {"x":599,"y":596,"time":1411732873180},
  {"x":596,"y":596,"time":1411732873188},
  {"x":592,"y":597,"time":1411732873196},
  {"x":589,"y":599,"time":1411732873204},
  {"x":585,"y":599,"time":1411732873212},
  {"x":582,"y":600,"time":1411732873220},
  {"x":579,"y":600,"time":1411732873228},
  {"x":577,"y":602,"time":1411732873236},
  {"x":574,"y":603,"time":1411732873242},
  {"x":572,"y":605,"time":1411732873258},
  {"x":571,"y":605,"time":1411732873266},
  {"x":569,"y":607,"time":1411732873274},
  {"x":567,"y":608,"time":1411732873282},
  {"x":565,"y":609,"time":1411732873290},
  {"x":560,"y":611,"time":1411732873298},
  {"x":557,"y":613,"time":1411732873306},
  {"x":556,"y":613,"time":1411732873314},
  {"x":553,"y":615,"time":1411732873320},
  {"x":552,"y":616,"time":1411732873331},
  {"x":551,"y":616,"time":1411732873336},
  {"x":550,"y":617,"time":1411732873345},
  {"x":548,"y":620,"time":1411732873352},
  {"x":548,"y":622,"time":1411732873360},
{"x":546,"y":623,"time":1411732873368},
{"x":545,"y":627,"time":1411732873376},
{"x":545,"y":629,"time":1411732873384},
{"x":544,"y":633,"time":1411732873392},
{"x":543,"y":636,"time":1411732873400},
{"x":542,"y":642,"time":1411732873406},
{"x":540,"y":647,"time":1411732873414},
{"x":539,"y":653,"time":1411732873422},
{"x":538,"y":657,"time":1411732873430},
{"x":537,"y":659,"time":1411732873438},
{"x":536,"y":664,"time":1411732873446},
{"x":535,"y":669,"time":1411732873454},
{"x":535,"y":670,"time":1411732873462},
{"x":534,"y":674,"time":1411732873470},
{"x":534,"y":675,"time":1411732873478},
{"x":533,"y":680,"time":1411732873486},
{"x":532,"y":684,"time":1411732873492},
{"x":532,"y":689,"time":1411732873500},
{"x":532,"y":690,"time":1411732873508},
{"x":532,"y":691,"time":1411732873516},
{"x":533,"y":693,"time":1411732873524},
{"x":535,"y":695,"time":1411732873540},
{"x":535,"y":696,"time":1411732873556},
{"x":536,"y":696,"time":1411732873564},
{"x":537,"y":697,"time":1411732873570},
{"x":539,"y":698,"time":1411732873578},
{"x":540,"y":699,"time":1411732873594},
{"x":542,"y":700,"time":1411732873602},
{"x":544,"y":700,"time":1411732873610},
{"x":549,"y":701,"time":1411732873618},
{"x":550,"y":701,"time":1411732873626},
{"x":553,"y":701,"time":1411732873634},
{"x":556,"y":701,"time":1411732873642},
{"x":559,"y":701,"time":1411732873650},
{"x":562,"y":701,"time":1411732873656},
{"x":565,"y":701,"time":1411732873664},
{"x":568,"y":701,"time":1411732873672},
{"x":571,"y":702,"time":1411732873680},
{"x":572,"y":702,"time":1411732873688},
{"x":576,"y":702,"time":1411732873696},
{"x":579,"y":702,"time":1411732873705},
{"x":587,"y":702,"time":1411732873713},
{"x":591,"y":702,"time":1411732873720},
{"x":594,"y":702,"time":1411732873728},
{"x":601,"y":702,"time":1411732873736},
{"x":606,"y":702,"time":1411732873742},
{"x":610,"y":702,"time":1411732873750},
{"x":615,"y":702,"time":1411732873758},
{"x":618,"y":702,"time":1411732873766},
{"x":622,"y":702,"time":1411732873774},
{"x":627,"y":702,"time":1411732873782},
{"x":630,"y":702,"time":1411732873790},
 {"x":632,"y":702,"time":1411732873798},
 {"x":636,"y":702,"time":1411732873806},
 {"x":639,"y":702,"time":1411732873814},
 {"x":642,"y":702,"time":1411732873820},
 {"x":642,"y":701,"time":1411732873828},
 {"x":644,"y":701,"time":1411732873836},
 {"x":645,"y":701,"time":1411732873852},
 {"x":646,"y":701,"time":1411732873868},
 {"x":648,"y":700,"time":1411732873876},
 {"x":649,"y":700,"time":1411732873884},
 {"x":651,"y":700,"time":1411732873892},
 {"x":653,"y":700,"time":1411732873900},
 {"x":656,"y":700,"time":1411732873906},
 {"x":657,"y":700,"time":1411732873914},
 {"x":658,"y":700,"time":1411732873922}],
[{"x":524,"y":741,"time":1411732874446},
 {"x":526,"y":741,"time":1411732874462},
 {"x":527,"y":741,"time":1411732874470},
 {"x":529,"y":741,"time":1411732874478},
 {"x":532,"y":740,"time":1411732874484},
 {"x":537,"y":740,"time":1411732874492},
 {"x":539,"y":740,"time":1411732874500},
 {"x":543,"y":740,"time":1411732874508},
 {"x":548,"y":740,"time":1411732874516},
 {"x":550,"y":740,"time":1411732874524},
 {"x":558,"y":740,"time":1411732874532},
 {"x":567,"y":740,"time":1411732874540},
 {"x":575,"y":740,"time":1411732874548},
 {"x":580,"y":740,"time":1411732874556},
 {"x":587,"y":740,"time":1411732874564},
 {"x":591,"y":740,"time":1411732874570},
 {"x":599,"y":740,"time":1411732874578},
 {"x":602,"y":740,"time":1411732874586},
 {"x":610,"y":739,"time":1411732874594},
 {"x":615,"y":739,"time":1411732874602},
 {"x":621,"y":738,"time":1411732874610},
 {"x":628,"y":738,"time":1411732874618},
 {"x":633,"y":738,"time":1411732874626},
 {"x":638,"y":738,"time":1411732874634},
 {"x":646,"y":738,"time":1411732874642},
 {"x":652,"y":738,"time":1411732874650},
 {"x":655,"y":738,"time":1411732874656},
 {"x":661,"y":738,"time":1411732874664},
 {"x":664,"y":738,"time":1411732874672},
 {"x":671,"y":738,"time":1411732874680},
 {"x":676,"y":738,"time":1411732874688},
 {"x":681,"y":739,"time":1411732874696},
 {"x":686,"y":740,"time":1411732874704},
 {"x":692,"y":741,"time":1411732874712},
 {"x":697,"y":742,"time":1411732874720},
 {"x":702,"y":742,"time":1411732874728},
 {"x":705,"y":742,"time":1411732874734},
{"x":706,"y":742,"time":1411732874742}]]
```

F. HWRT Handbook

The Python package `hwrt` can be installed via pip:

```
# pip install hwrt
```

The toolkit requires a configuration file `/.hwrtrc` that contains your projects root folder and the name of your neural network toolkit:

```
root: /home/moose/Downloads/write-math
nntoolkit: programname
```

After that, it can be checked via command line if the installation worked:

```
$ hwrt --version
hwrt 0.1.150
```

The project development hosted on `https://github.com/MartinThoma/hwrt`.

`hwrt 0.1.X` works in your projects root folder. Inside of `project root` it looks for the following folders

- `raw-datasets`: Flat folder that contains one `info.yml` and the raw datasets as `.pickle` files. Pickle is the standard way to serialize objects in Python.
- `preprocessed`: Folder that contains other folders. Each folder describes one specific way to preprocess data as well as the raw data source within a `info.yml` and contains the preprocessed files as `.pickle` files.
- `feature-files`: Folder that contains other folders. Each folder describes a set of features and the data source that should be used within a `info.yml`. The feature-files are created in those folders in the `.pfile` format.
- `models`: Folder that contains other folders. Each folder contains an `info.yml` that describes the feature file data source, the model and how to train the model.

The YAML configuration file for the preprocessing queue, `info.yml`, looks like this:

```
data-source: archive/raw-datasets/2014-08-26-20-14-handwriting_datasets-raw.pickle
queue:
  - RemoveDuplicateTime: null
  - StrokeConnect:
      - minimum_distance: 10
  - ScaleAndShift:
      - max_width: 1.0
      - max_height: 1.0
      - center: true
  - SpaceEvenlyPerStroke:
      - kind: linear
      - number: 20
  - ScaleAndShift:
      - max_width: 1.0
      - max_height: 1.0
      - center: true
```

The queue is ordered and can contain duplicate elements. All features that are classes in hwrt/preprocessing.py can be used in this list. The data-source is relative to the project root folder.

The YAML configuration file for features, info.yml, looks like this:

```
data-source: archive/preprocessed/c2
data-multiplication:
  - Multiply:
      - nr: 1
features:
  - ConstantPointCoordinates:
      - strokes: 4
      - points_per_stroke: 20
      - fill_empty_with: 0
      - pen_down: false
  - ReCurvature:
      - strokes: 4
  - Ink: null
  - StrokeCount: null
  - AspectRatio: null
```

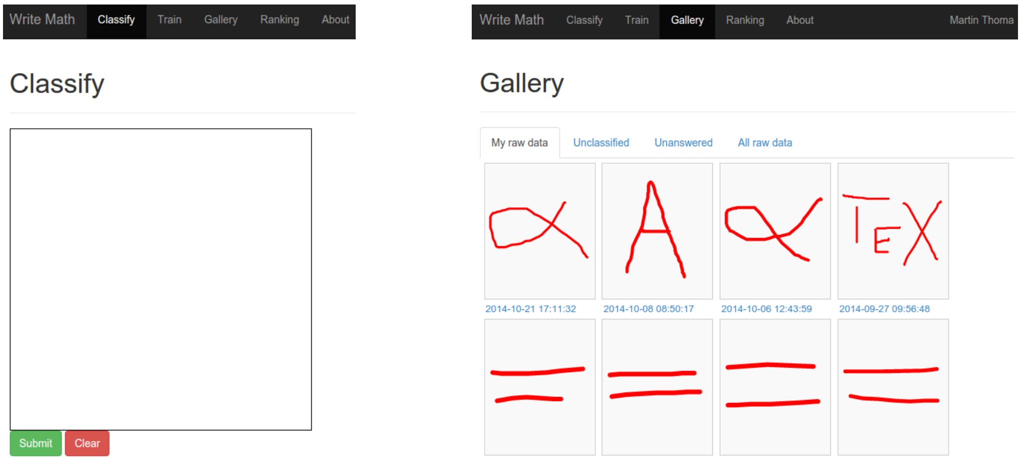
All features that are classes in hwrt/features.py can be used in this list.

The model info.yml looks like this:

```
data-source: archive/feature-files/c2
training: '{{nntoolkit}} train --epochs 1000 --learning-rate 0.1
    --momentum 0.1
    {{training}} {{validation}}
    {{testing}} < {{src_model}} > {{target_model}} 2>> {{target_model}}.log'
model:
    type: mlp
    topology: 167:500:500:369
```

The training parameter makes use of templates. {{nntoolkit}} gets replaced by the string that was specified in ~/.hwrtrc, {{training}} gets replaced by the training pfile, {{validation}} gets replaced by the validation pfile and {{testing}} gets replaced by the testing pfile. The training algorithm looks for model-[number].json and replace {{src_model}} by the latest model path. {{target_model}} gets replaced by model-[number+1].json.

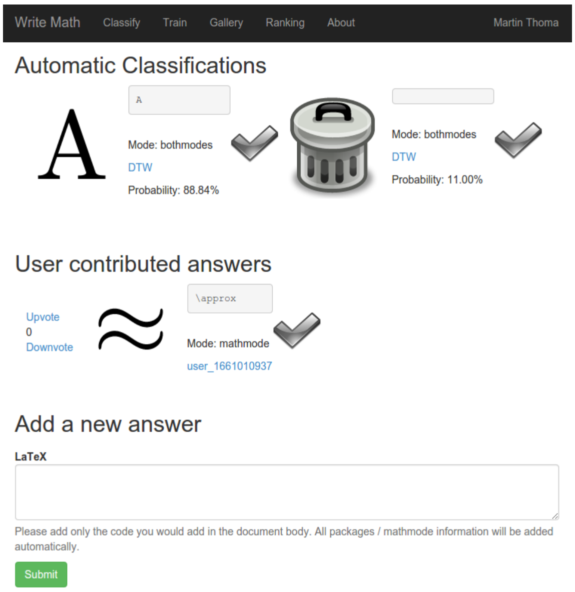
G. Website



(a) Webpage where users can record their handwriting

(b) Gallery page where the user can see what was drawn and what is unclassified

Figure G.4.: *Screenshots of different pages of* `write-math.com`



Automatic Classifications

A

Mode: bothmodes

DTW

Probability: 88.84%

Mode: bothmodes

DTW

Probability: 11.00%

User contributed answers

\approx

Mode: mathmode

user_1661010937

Add a new answer

LaTeX

Please add only the code you would add in the document body. All packages / mathmode information will be added automatically.

Figure G.5.: *Page on which the user can see a recording and which symbols were suggested by automatic classifiers as well has human classifiers. The human classifications can get accepted and rated.*