# On the Security of reCAPTCHA

Bachelor-Thesis by Benjamin Milde from Darmstadt July 2010



Darmstadt University of Technology Department of Computer Science Minimizing Cryptographic Assumptions (MiniCrypt) On the Security of reCAPTCHA

Bachelor-Thesis by Benjamin Milde from Darmstadt Supervised by: Dr. Marc Fischlin and Paul Baecher Day of submission: 31. July 2010

## Erklärung zur Bachelor-Thesis

Hiermit versichere ich, die vorliegende Bachelor-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 31. Juli 2010

(B. Milde)

#### Abstract

CAPTCHAs are a popular method for stopping automated attacks and to reduce spam on websites. The idea is to use a puzzle, that only a human being can solve, but not automated programs. Particularly text CAPTCHAs, that ask a user to decipher distorted text in an image, are widely deployed on websites and users are accustomed to this type of CAPTCHA. reCAPTCHA is one of the biggest provider of text based CAPTCHA technology and is said to be one of the most secure. Current optical character recognition methods, especially traditional segmentation methods, work unreliably on distorted words from reCAPTCHA. In this bachelor thesis, the security of reCAPTCHA is analyzed in the presence of a custom automated solver capable of holistic word recognition. Specialized preprocessing is needed for one of reCAPTCHA's recent variants, and proposed on the basis of a machine learning algorithm. Existing object recognition methods are modified and extended to work fast in a search space of about 22'000 words. It is experimentally shown that this method works well with all recent variants of reCAPTCHA's challenges and achieves non-trivial recognition rates.

1	Introduction	4
2	Background         2.1       Principles         2.2       Used terminology	
3	Related work         3.1       Automated solvers         3.2       Human aspects in word recognition         3.3       Holistic word recognition	7
4	<ul> <li>Analysis of reCAPTCHA</li> <li>4.1 Acquisition of sample data</li> <li>4.2 Sample CAPTCHAs</li> <li>4.3 Analysis of third generation CAPTCHAs</li> <li>4.3.1 Additional distortions</li> <li>4.3.2 Manual categorization of verification words after edge detection</li> <li>4.3.3 Word distributions of the CAPTCHAs</li> <li>4.3.4 Estimation of dictionary size used by reCAPTCHA</li> <li>4.3.5 Analysis of individual collisions</li> <li>4.4 Trivial solving rate</li> <li>5 Fourth generation of CAPTCHAs</li> </ul>	8 9 10 10 11 11 13 13
5	Preprocessing         5.1       General Idea         5.1.1       Problem formulation         5.2       Prerequisites         5.2.1       Training set         5.2.2       Training set	14 14 14
	<ul> <li>5.2.2 Estimation of the ellipse center</li> <li>5.2.3 Features</li> <li>5.3 Classifier</li> <li>5.4 Classifier cascade</li> <li>5.4.1 Additional features</li> <li>5.5 Results</li> </ul>	16 16 18 18
	Holistic word recognition         6.1 Motivation         6.2 Shape contexts         6.2.1 Building a shape context         6.3 Discrimination of shapes         6.3.1 Comparison         6.3.2 Appearance similarity         6.3.3 Generalized shape context         6.4 Database         6.5 Search algorithms         6.5.1 Naive search algorithm         6.5.2 Better-half-search algorithm         6.5.3 Pruning with single characters	24 24 25 25 26 26 27 27 28 28 28 28 30
	7.1 Practical considerations	<b>31</b> 34
	Conclusion Acknowledgements	36 37
3	Acknowledgements	57

#### 1 Introduction

A CAPTCHA is an automatic test, that can be taken over the internet and is used to differentiate humans from machines. The acronym stands for "Completely Automated Public Turing test to tell Computers and Humans Apart" and it was coined by Luis von Ahn et al. in the year 2000 [38].

A CAPTCHA should be easy to solve by a human and should be difficult for a computer program. This is a reverse Touring Test [4]: a computer program decides if it deals with a human or a computer [21].

CAPTCHAs have a variety of applications, most notably to help prevent spam [38]. Bogus comments are being submitted by automated bots on websites, malicious users create thousands of accounts on free e-mail services to spread spam. Online-Polls are exploited by automated bots so that their outcome is manipulated. By using a CAPTCHA it is much harder to automate such tasks: if the website can reliably detect that a computer program is using it and not a human, it can decide to deny its service to computer programs.

Suitable unsolved Artificial Intelligence (AI) problems are used to build CAPTCHAs [3]. The security of a CAPTCHA is based on the assumption, that the underlying AI problem is one that computers cannot yet perform. The recognition problem of characters and words from images under clutter and distortions is often used for CAPTCHAs. Even though Optical Character Recognition (OCR) has a long tradition and predates electronic computers [28], it is still the case that humans are considered to be significantly better at recognizing words [25], particularly when they are distorted and noisy.

The recognition of single and isolated characters on the contrary is seen as a solved AI problem and current recognition algorithms can have a better accuracy than humans [14]. Segmentation into single characters is usually associated with OCR: if the characters of a word are only narrowly separated or not at all, it becomes very hard to segment them with an automated program. Studies have shown that computers are better at solving the recognition problem than the segmentation problem [15, 17].

reCAPTCHA is one of the most widely used CAPTCHAs on websites. Up to this date, it is based on the word recognition problem. Words from scanned books and newspapers are used [2], most of them are older so that they are subject to an aging process that has degraded, smudged and distorted the words. They can also be misaligned by the scanning process and could be printed in a variety of typefaces of which many could be rarely used today. The words used for the test have characters that are most of times not separated at all or leave very thin spaces between the characters. In addition to this, they are also distorted artificially to make the AI-problem of recognizing these words even harder.

By typing in two of this distorted words correctly, the user proves that he is human. Figure 1 shows a screenshot of an example CAPTCHA challenge from reCAPTCHA. The CAPTCHAs used by reCAPTCHA change from time to time, the screenshot was taken in early June 2010.



Figure 1: Example for a CAPTCHA challenge from reCAPTCHA (early June 2010)

reCAPTCHAs website reportes that over 30 million of such challenges are served every day (as of July 2010). The integration of reCAPTCHA into a website is free and packages for various web programming languages exist, making it easy for webmasters to adopt the technology. It is widely deployed and the reCAPTCHAs website currently states that over 100.000 websites are using reCAPTCHA. Very popular websites like Facebook, Twitter and StumbleUpon are using reCAPTCHA since at least 2007 [7].

Because reCAPTCHA is a very popular CAPTCHA, as outlined above, it is also an interesting target for an academic security analysis. The central question in analyzing the security of a CAPTCHA is whether it is possible to build an automated software solver that can solve a non trivial fraction of the challenges, thereby undermining the premise that the CAPTCHA is hard to solve by computers.

In this bachelor thesis, I present such an automated software solver and show that it succeeds to solve a non-trivial fraction of challenges for each of three different types of word CAPTCHAs deployed by reCAPTCHA in the last year. The solver does not use classical OCR algorithms, it also does not depend on some other OCR system and it is capable of solving between about 5% to up to 12.7% percent of the challenges, depending on the tested CAPTCHA generation. For the current one deployed by reCAPTCHA at the time of publishing this thesis, a solving rate of up to 11.6% could be achieved. The problem of segmentation is avoided by a proposed holistic recognizer, that tries to solve the CAPTCHAs on

a word basis. For this an approach for object recognition, called shape context [8] is used and an algorithm is presented that makes it possible to apply the concept to a similarity search in a large precomputed dictionary.

For the generation of CAPTCHAs deployed by reCAPTCHA between January and July 2010 suitable preprocessing is needed, because additional elliptic shapes are placed into the image to distract recognizers. For this I propose to use a specialized approach with a supervised learning algorithm to filter the CAPTCHA on a per pixel basis.

#### 2 Background

CAPTCHAs are also called Human Interaction Proofs (HIPs) in common literature. Initially, a CAPTCHA implied a system for which the generator is public (for example it is open source), as the "P" in CAPTCHA stands for public, whereas a HIP does not have this notion. However, as of today the terms CAPTCHA and HIP are used as synonyms. The term CAPTCHA is the preferred one in new publications and is well established. It is now also used for systems without a public generator [2].

#### 2.1 Principles

CAPTCHAs should satisfy three basic principles [20]. The tests must be

- 1. Easy for humans to pass.
- 2. Easy for a tester machine to generate and grade.
- 3. Hard for a software robot to pass. The only automaton that should be able to pass a CAPTCHA is the one generating the CAPTCHA.

#### 2.2 Used terminology

If the CAPTCHA system allows it, a small fraction of the tests can be solved by supplying the same solution to every challenge. The CAPTCHA is then solved at a *trivial solving rate*. It is  $\frac{1}{n}$ , if there are *n* possible solutions to the CAPTCHA and each solution is equally likely to be used as a challenge. It is widely agreed that a trivial solving rate for a good CAPTCHA should be less than 0.01% [16].

An adversary  $A_t$  is trivial, if he can only solve a fraction of the CAPTCHAs with less than or equal the trivial rate. An adversary  $A_h$  is partially-human, if he leverages the recognition of the CAPTCHA to a human. Such attacks exploit the first principle, that the CAPTCHA must be easily solvable by a human. An adversary  $A_s$  is a strong adversary, if he attacks design-flaws in the third principle and is able to pass the CAPTCHA computationally at a solving rate significantly above the trivial one. It is widely agreed (for example [2, 53]) that a CAPTCHA is effectively broken if there exists an  $A_s$  that can solve the challenges at a rate higher than 5%. A commonly accepted goal for a good CAPTCHA is that an adversary (without being partially human) should not be able to achieve a success rate of higher than 0.01% for passing the challenges, but that the human success rate should be at least 90% [16].

A *text CAPTCHA* shall be any CAPTCHA that uses the underlying AI-problems character recognition and segmentation. A text CAPTCHA is *segmentation resistant*, if it is difficult to segment the CAPTCHA into its individual characters with known character segmentation algorithms. Current research [1] provides further evidence that if a text CAPTCHA can be segmented into its individual characters, it is effectively broken. Thus, to satisfy the third principle above, a good text CAPTCHA should be segmentation resistant. Note that there is no way to prove segmentation resistance, it can only be shown empirically by testing certain OCR softwares or segmentation algorithms. Furthermore a text CAPTCHA shall also be a *word CAPTCHA*, if it uses words from natural language for its challenges.

#### 3 Related work

This section focuses mainly on work related to text CAPTCHAs, because reCAPTCHA is also a text CAPTCHA.

The first practical text-CAPTCHA system was described in a patent filed in 1998 by Compaq Computer Corp. [37]. Ahn et al. introduced the notion of a CAPTCHA in 2000 [38], followed by a formal description in 2003 [3]. They are also said to be the creators of EZ-Gimpy and Gimpy [9], two early text CAPTCHAs developed for Yahoo.

Nagy et al. showed in 1999 that the performance of Optical Character Recognition (OCR) programs of the time were inferior to the reading ability of a seven year old child [49]. Coates et al. used these results in 2001 to describe a text CAPTCHA based on words of low and degraded quality [21].

Simard et al. argued in 2003 [52] that the task of recognition, at least of printed isolated characters, is solved. The task of segmentation however, is still a difficult AI-problem and text CAPTCHAs should focus on making the segmentation difficult. Basically, they described the notion of a *segmentation resistant* text CAPTCHA.

Baird et al. introduced ScatterType in 2005, a CAPTCHA that uses words that are artificially created in way to resist traditional character segmentations techniques from OCR [5], so that the CAPTCHA is segmentation resistant. Ahn et al.

introduced reCAPTCHA in 2008 [2], a CAPTCHA that uses words from old books and helps to digitalize them. Though not explicitly stated, the words appear to be segmentation resistant (see Section 4.3). It quickly gained popularity and became one of the most used CAPTCHA-systems world-wide, as outlined in the introduction section.

General and recent overviews in the area of OCR are given by Fujisawa et al. (2008) [25] and Cheriet et al. (2007) [18], summarizing the advancements in the field in last decades.

#### 3.1 Automated solvers

Since its introduction, text CAPTCHAs have been the most widespread CAPTCHAs. Unsurprisingly, a large part of work that deals with the security of CAPTCHAs focuses on text CAPTCHAs.

In 2003, a landmark paper in this area by Mori et al. [46] describes an automated software solver for the EZ-Gimpy and Gimpy CAPTCHA, two difficult and distorted text CAPTCHAs for that time period. Gimpy was solved with a success rate of 33% and the easier variant EZ-Gimpy CAPTCHA with 92%.

They used the idea of *shape contexts*, a mathematical concept for the similarity of shapes, that they developed earlier in 2001 [43,45]. By interpreting letters and whole words as shapes and defining the recognition as a search for a similar shape the EZ-Gimpy and Gimpy there successfully broken by shape context comparisons. The idea was further developed to improve the robustness of shape comparisons and to extended the idea [8,33,34,44]. This work is the foundation for the recognizer in this bachelor thesis.

Shortly after, Moy et al. [48] developed techniques to deal better with the distortions in EZ-Gimpy to improve recognition rates to 99% and also breaking the very distorted variant Gimpy-r with a success rate of 78%. For EZ-Gimpy a holistic word recognition approach was also chosen and for Gimpy-r individual characters were recognized, because they were easy to segment.

Chellapilla et al. analyzed and broke several CAPTCHAs using a different approach, by applying concepts of machine learning to both segment and recognize characters [17]. Their universal solver showed a good response to most analyzed CAPTCHAs and a after an individual learning phase final success rates ranging from slightly below 5% to above 66% could be reached. One of the key conclusions is that once the segmentation problem is solved, solving the CAPTCHA becomes a pure recognition problem and it can trivially be solved using machine learning algorithms [17]. They also suggest that the security of a CAPTCHA is generally increased by making the segmentation problem more difficult.

In 2005, Chellapilla et al. [14] showed that adding further distortions to individual letters does not necessarily make the problem of recognizing them more difficult for computers. On the contrary, the results show that cases can be constructed where computers (using neural networks) are even better than humans. Table 1 shows this exemplary.

Characters under typical distortions	Recognition rate of computers
54220	pprox 100%
RRFBD	96+%
6 T & C B	100%
$D_{1}^{2} = \frac{3}{2}$	98%
创新做具家	≈ 100%
運行運動	95+%

 Table 1: Recognition rates for individual characters with machine learning (neural networks) for different forms of distortions, as shown exemplary. All data taken from [14, 59].

A number of additional security analysis of text CAPTCHAs appeared that use segmentation into characters and character recognition as their primal method for a software solver [1,29,59]. This gives further evidence that once a CAPTCHA can be segmented automatically and reliably into its individual characters, it is trivial to break it.

In December 2009, Wilkins "Strong CAPTCHA guidlines" [58] got public and caused quite a stir regarding an automated software solver for reCAPTCHA. It was meant to be a guideline to make text CAPTCHAs more secure and Wilkins tested reCAPTCHA, among others, with the open source OCR software Ocrupus [11], that uses segmentation for the recognition process. However the methods of measurements were not outlined quite clear and no timings were given. Motoyama et al. [47] showed in a recent publication an economic perspective on CAPTCHAs, arguing that partially-human adversaries are used in practice to circumvent CAPTCHAs and that a whole industry and market arised that sells human CAPTCHA solving services in bulk rates<sup>1</sup>. Additionally, Motoyama et al. could test Wilkins solver on the second generation CAPTCHAs more accurately. They estimated that it could solve 18% of the second generation CAPTCHAs, with an average time of 105 seconds on a Core2Duo with 2.13 GHz, and they optimized it to 12 seconds with a slightly smaller accuracy of 17%.

#### 3.2 Human aspects in word recognition

Human word recognition has been extensively studied by psychologist throughout the past century [13, 24, 27, 50, 57]. Two different theories emerged, the holistic theory suggests that humans recognize whole words at once, whereas hier-archical theories suggest that humans segment words into individual parts or letters.

Frost et al. [24] suggests that both theories are true to some extend and humans most likely use a combination of both to recognize words. This theory is based on observations with dyslexia, a condition that impairs the ability to read; patients with surface dyslexia [6] read frequent words as if they were new to them and patients with deep dyslexia [51] are still able to read familiar words, but are unable to read new words by individual letters. This gives evidence that humans have the ability to segment words in characters and also recognize whole words at once.

The *word-superiority effect* [13], which states that humans are able to recognize certain words faster than individual characters, is further evidence that a holistic recognition process plays a strong role in human word recognition. [36]

#### 3.3 Holistic word recognition

Up to this date, the role of holistic word recognition has not been intensively studied in connection with automated CAPTCHA solvers. Mori et al. and Moy et al. both used the idea for EZ-Gimpy [46, 48], however EZ-Gimpy used a dictionary of 561 words. The Oxford English Dictionary lists about 500'000 words and an average educated person knows about 20'000 words [30], allowing word CAPTCHAs that use only (known) English words with far bigger dictionary sizes.

However, holistic word recognition has been studied in the context of handwriting recognition, inspired by the results in cognitive psychology. Madhvanath et al. [40] studied the role of holistic word recognition for handwriting recognition. Koerich et al. gave an overview in 2003 of the different techniques used for holistic hand writing recognition at that time [32]. Most approaches use Hidden Marcov Models (HMMs), with varying sizes for the wordlists; the approach by 2001 Marti et al. [41] showed a recognition rate of about 60% for vocabulary sizes ranging from 2703 to 7719 words. Vinciarelli et al. [55] reached a recognition rate of 85% with a lexicon containing 50'000 words in 2003.

#### 4 Analysis of reCAPTCHA

The main novel idea of reCAPTCHA is to use the superior ability of humans at reading words for digitalizing purposes [2]. That is why the CAPTCHA-test consists of two distorted words that should be recognized: One of them is the *verification word*, for which the correct answer is known and that is being used as the actual CAPTCHA-test. The other word is the *read word*, a new word that comes from a book that should be digitalized and is hard to recognize by computers. With statistical tests the preferred answer of humans to the read word is evaluated and used as the solution to the recognition problem. Upon evaluation of an answer, the read word could be also used as a new verification word.

This procedure has an economic value, as it can be used to make a human OCR machine with low operation costs. Ahn et al. [2] stated that it has a higher accuracy than professional human transcribers, with a transcription rate over 99% at word level. reCAPTCHA was bought by Google in September 2009 [26] and is known to use the economic value of reCAPTCHA for the Google Books and Google News Archive Search project, in which old books and newspapers that are not available in digital form are transcribed.

The solution to the read word is not known by reCAPTCHA; so in order to pass the CAPTCHA, only the verification word must be correct. In fact, in my empirical experiments it sufficed to just use only a correct verification word as a solution. If two words are supplied as a solution to the CAPTCHA, the verification word must be on the same position as shown in the image, otherwise the solution will not be accepted.

No distinction between upper and lower case characters is made, so that "THESIS" is also a valid solution to the verification word "thesis". The solution to the verification word is also accepted, if it has a Levenshtein distance of one to the solution known by reCAPTCHA, this allows an "off by one" error from the user to be still accepted. It is also explicitly stated on the reCAPTCHA wiki for frequently asked questions:

<sup>&</sup>lt;sup>1</sup> In 2010, the market value of solving 1000 text CAPTCHAs is approximately 1–2\$

#### Why is reCAPTCHA accepting incorrect words?

[...] The read word is not graded (since the server is using human guesses to figure out the answer). As such, this word can be entered incorrectly, and the CAPTCHA will still be valid. Each read word is sent to multiple people, so incorrect solutions will not affect the output of reCAPTCHA.

On the verification word, reCAPTCHA intentionally allows an "off by one" error depending on how much we trust the user giving the solution. This increases the user experience without impacting security. reCAPTCHA engineers monitor this functionality for abuse.

According to the founders of reCAPTCHA [2], words are being used as verification words that two state-of-the-art OCR programs failed to recognize, but humans tend to recognize well. This can explain why it is hard to segment those words into characters: Most likely the OCR programs used by reCAPTCHA use segmentation into characters and character recognition as their primal method of word recognition, so that they fail particularly on words that are hard to segment, which then in turn are used as a CAPTCHA.

#### 4.1 Acquisition of sample data

The generator for the CAPTCHAs of reCAPTCHA is unfortunately not publicly available, so samples have to be collected manually for an analysis. For this I created a website<sup>2</sup> that integrates reCAPTCHA following the installation instructions from reCAPTCHA. The PHP-code provided by reCAPTCHA for this can be simply edited to record the CAPTCHAs after they have been validated by the reCAPTCHA server, so that samples that reCAPTCHA accepts as correct solution can be collected on the server that hosts the website.

Because only the verification words are relevant to solve the CAPTCHA, it is important to collect the data in such a way that one can distinguish between verification word and read word. For this, we can use the observation that the read word can in fact be any word, while the solution will be still accepted. For every challenge we want to collect, we choose the same bogus token for the suspected read word, for example "42" and fill in the right word for the verification word. The reCAPTCHA server then verifies our solution. If we used the bogus token accidentally for the verification word, the solution is not accepted. But if our guess turned out to be right and our verification word matches to the CAPTCHA, the solution will be accepted. Depending on the position of the the bogus token in our solution, we also know the position of read and verification word in the image, along with a solution for the verification word.

According to my data (2000 CAPTCHAs), the distribution of the verification words position is approximately 50% on both sides of the CAPTCHA. After some practice and feedback from the reCAPTCHA server, I learned to guess the position of the verification word accurately and noted that verification words and read words have some distinct properties (see also Section 4.3).

#### 4.2 Sample CAPTCHAs

For security reasons, reCAPTCHA changed at least three times its major CAPTCHAs since its introduction. The basic principle remains the same, the CAPTCHA is based on the word recognition problem. However, the way the CAPTCHAs are artificially distorted changed.

Major reCAPTCHA generations are the sort of CAPTCHAs that most websites show for a longer period of time. Figure 2 shows examples for the three major CAPTCHA generations since reCAPTCHAs introduction.

reCAPTCHA also seems to experiment with other CAPTCHAs sometimes, which I will call minor reCAPTCHAs. Figure 3 shows examples for some other CAPTCHAs that there collected during my analysis. With the second generation CAPTCHAs, there was an additional security measurement in place, so that after around 40-50 failed attempt to solve it a different and more difficult CAPTCHA (see Figure 3(c)) would be shown. This could not be observed with the third generation reCAPTCHAs, but it could just be that this security measurement is in place after many more failed attempts now.







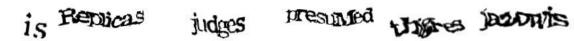
(a) First generation, early 2008

(b) Second generation, until Decem- (c) Third generation, since January ber 2009

2010

Figure 2: Examples for the three major reCAPTCHA generations, as observed from Darmstadt/Germany, until June 2010

http://www.cdc.informatik.tu-darmstadt.de/~b\_milde/recaptcha/



(a) At the end of December 2009 (b) End of December 2009, capital (c) Very difficult CAPTCHA, after reCAPTCHA experimented with capital letters at the beginning of the verification word

letter in the middle of the verification word

too many second generation CAPTCHAs had been solved wrong

## reported tornados

(d) June 2010: A different CAPTCHA is shown on recaptcha.net for the time being, but other websites show the third generation CAPTCHA.

Figure 3: Examples for minor reCAPTCHAs

#### 4.3 Analysis of third generation CAPTCHAs

The analysis in this bachelor thesis focuses on the third generation reCAPTCHAs, as shown exemplary in Figure 2(c). The central additional security measure for this third generation, compared to the previous ones, is the use of a dominant additional distortion, that has the rough shape of an ellipse.





(a) 42 dunking





(c) croaked 42



(b) 42 smocking



(d) salaams 42

Figure 4: More examples for third generation reCAPTCHAs alongside with the solution that was used to solve the CAPTCHA and that got verified by the reCAPTCHA server. Thus, all words tagged with bogus token "42" are read words.

The verification words collected in April and May 2010 are nearly all words from the English language, but sometimes also names for locations or people. I could also observe that none of the collected verification words had a capital letter. The read words on the other hand are quite different, capital letters and also numbers can be observed.

All CAPTCHAs can be easily segmented into single words, by searching the biggest white space near the center of the CAPTCHA. This is because all third generation CAPTCHAs sufficiently separate read and verification word by enough free space. However nearly all collected verification words showed words that are very difficult to segment into individual characters, because most letters of the word are in touch with each other. Simple segmentation algorithms like the connect component analysis (CCA) [19], that could yield good results on printed text with good quality, would most likely fail on all of the verification words of reCAPTCHA to give good results. CCA tries to identify character boundaries by spaces between the characters and is based on the observation that in most printed text the characters do not touch. If however most characters are also connected, because they touch each other - this is the case with the verification words - this simple algorithm does not work for obvious reasons.

In principle, reCAPTCHA is implicitly segmentation resistant through this construction; that is, it is most likely so for the two OCR softwares that are used to select the challenges. But, as with CAPTCHAs that are explicitly designed to be segmentation resistant, testing against a certain OCR software or a segmentation algorithm is no guarantee that the CAPTCHA is actually segmentation resistant. There could simply be a better segmentation algorithm that succeeds to segment the words into its characters. But with the used concept, reCAPTCHA has the advantage to additionally

incorporate any segmentation algorithm into the selection process that performs better; by sorting out all verification words that can be segmented with the better algorithm, the challenges remain segmentation resistant.

It is worth noting, that the read words again are different and not always segmentation resistant. Some of them have in fact characters that are well spaced and thus should be easy to segment and recognize for modern OCR software. However, solving (or not solving) the read word is irrelevant to passing the CAPTCHA. This stresses that a security analysis on reCAPTCHA should focus on the verification words.

The fonts used for the verification words share a higher similarity between the words, based on my observations, as it is the case with the read words. Also I can conclude that the majority of the words uses fonts with serifs. This makes perfect sense: the words come from printed books and newspapers, where serifed fonts are used today and have been used in the past for body text, because they are considered easier to read than sans-serif fonts for this purpose [42].

The words from the third generation CAPTCHAs look artificially deformed, most likely some randomized transformations have been applied to the words by reCAPTCHA. However, the deformations seems to have a stronger influence on the relative y-axis of the word (compared to the height of the word), than on the relative x-axis (compared to the width). This is reasonable to maintain readability for humans.

#### 4.3.1 Additional distortions



(a) Another example image for a third generation reCAPTCHA, collected in April 2010



(b) The verification word from the example on the left after upscaling and after Canny edge detection [12]. Here, the character "d" has been damaged by a close edge of the elliptic distortion.

Figure 5: Applying an edge detector on a verification word.

The additional distortions are distorted ellipses and we can observe them likewise for verification and read words. The areas under those ellipses are inverted, so that a part of the word is shown in white letters on the black elliptic shape and usually a part of the word is outside of the distortion, with normal black letters. If an edge detector [12] is applied to the image (see Figure 5), an additional edge is visible for the ellipse. The advantage of using edge detection is certainly that edges are detected likewise for the inverted part and the normal part of the word, we do not have to make a special distinction between both parts. If the edge of the ellipse comes close to edges of characters, then some edges from the original word can be damaged, for example see Figure 5(b) where the character "d" has been damaged though the additional edge.

Because the deployed Canny edge detector outputs only binary data, the original greyscale image had to be scaled to 200% of its original size so that the edges are more accurate. Applying edge detection in this form onto the CAPTCHA seems to make the problem of filtering the distortion easier, as only the outer edge of the ellipse has to be separated from the other edges as compared to inverting back the part under the ellipse. For the latter it would be fatal to invert different parts than the ellipse of the image, because that would introduce new distortions. However, a certain margin of error (for example 90% accuracy) would be tolerable, if we just filter the edges.

#### 4.3.2 Manual categorization of verification words after edge detection

I have manually analyzed 130 verification words after edge detection, collected in April 2010. They have been categorized in four broad categories and are meant to give an approximate picture about the severeness of distortions produced by the ellipses. See table 2 for an overview.

Category	%	Example edge image	Description
Slight deformation	44,6%		A big part of the ellipse clearly visible and
		0	enough space to most parts of the word.
		(any GMOUNS	
Medium deformation	43,9%		Ellipse goes through a substantial part of
		antimolec	the word or is generally nearer to the
		SINI FPCCS	word and it shows bigger deformation.
Strong deformation	6,9%		Difficult to fit an ellipse onto the addi-
		- A-A	tional edge or serve damage to the word
		DIWIMAR	because ellipse goes through the biggest
		5 - Bellico	part of the word.
Other	4,6%		The CAPTCHA does not have an addi-
		~ 0	tional distortion in form of an ellipse (is
		12073/120	a different kind of CAPTCHA).
		V	

Table 2: Severeness of distortions produced by the ellipses, exemplary categorization.

#### 4.3.3 Word distributions of the CAPTCHAs

In [2], Ahn et al. state that the distribution D of verification words is uniform, so that every word that is part of D has the same probability of being used as a CAPTCHA. This is important so that a trivial adversary can not pick the most common solution.

The distribution of the read words however is different: In a manual experiment, I counted the occurrence of the word "the", the most used word in the English language, in the read words as 11 times out of 100 CAPTCHAs. This suggests that the distribution of the read words could follow a similar distribution as the words in an English text. This also demonstrates that it is important to normalize the distribution of the verification words: if they would follow the same distribution as the read words, an trivial adversary that outputs "the" to all CAPTCHAs could solve 11% in my previous example.

This also means that the difficulty to solve the read words cannot be compared to the verification words, as an adversary could use the different probabilities of the distribution in the first case to his advantage.

#### 4.3.4 Estimation of dictionary size used by reCAPTCHA

Let *d* be the dictionary of reCAPTCHA, that is defined as all unique solutions to all possible verification words, ignoring lower and upper case. If the distribution *D* of a random word from *d* is uniform, it has the probability 1/|d| of being chosen as a verification word for a CAPTCHA.

It is interesting to know an estimate of |d|. For this, the number of collisions after solving *n* CAPTCHAs can be used. A word is a collision to the previous n - 1 CAPTCHAs, if they also contain the same word. After I obtained 1932 verification words in April and May 2010, the number of collisions was 44.

The probability of a collision for the *n*-th CAPTCHA is related to a variation of the birthday problem. In this variation the probability of a particular person to have the same birthday as one of the n other persons in the same room is asked. The probability for this is:

$$q(n) = 1 - \left(\frac{364}{365}\right)^r$$

This gives the following generalized formula, on a set of size *d*:

$$q(n;d) = 1 - \left(\frac{d-1}{d}\right)^n$$

The probability of a collision to at least one of the n - 1 verification words after n CAPTCHAs have been solved is then given by q(n - 1; d).

The expected value of collisions after n verification words and a dictionary of size d can now be derived from this probability:

$$E(\text{collisions after n verification words}) = \sum_{k=1}^{n} q(k-1;d) = \sum_{k=1}^{n} \left(1 - \left(\frac{d-1}{d}\right)^{k-1}\right) = d\left(\frac{d-1}{d}\right)^{n} + n - d$$

Under the assumption that the collected data from April and May 2010 is a collision outcome that is near the expected value of collisions and that the distribution D for reCAPTCHAs verification words is indeed uniform, d can be approximated by:

$$44 = d\left(\frac{d-1}{d}\right)^{1932} + 1932 - d$$

This formula can be solved numerically:  $d \approx 41'749$  words.

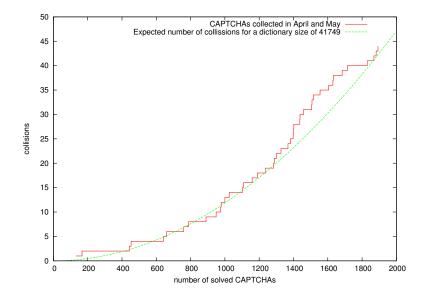


Figure 6: Collisions after solving n reCAPTCHAs, observed values and expected values for a dictionary size of 41'749

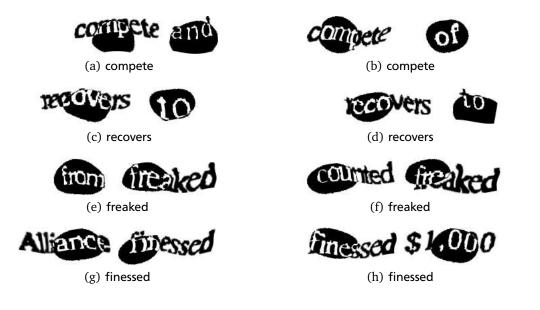


Figure 7: Individual collisions for the verification words.

Figure 6 shows the expected number of collision for this approximated dictionary size after *n* words and the real observed values through this experiment. The number stated by Ahn et al. in [2] for this is 100'000. The expected number of collisions for n = 1932 and a dictionary size of 100'000 unique solutions would be 18.5, approximately half the observed value. This has to be taken with a grain of salt: the assumptions made for the formulas could be wrong, particularly it could be that the outcome of the experiment is not near the expected value. Also the distribution of the verification words could be not fully uniform. Alternatively, the size of dictionary is indeed (or additionally) a bit different for the third generation CAPTCHAs as compared to the first generation in 2008, the publishing date of [2].

#### 4.3.5 Analysis of individual collisions

The individual collisions from the previous example are a good way to analyze the amount of variance the CAPTCHAs have even though they have the same solution. Some randomly selected examples are shown in Figure 7.

The elliptic distortions are every time in different positions and shapes, showing an expected amount of variety and are most likely computed every time a new CAPTCHA challenge is generated. The amount of variance in the deformation of words is surprisingly small between two CAPTCHAs of a collision, this is at least the case for the collisions in Figure 7. Also it is sometimes hard to tell if the words are actually generated from the same image source or from different images, because the font and appearance of the words is surprisingly similar; but that would also be true for a word that appears in different locations in the same book.

#### 4.4 Trivial solving rate

A trivial adversary  $A_t$  for reCAPTCHA could output the word "care" to each challenge. Under the assumption that the reCAPTCHA server would allow this and verifies each word, one would think that the chances are approximately  $\frac{1}{n}$  for each challenge to get the solution verified, where *n* is the size of reCAPTCHAs dictionary. However "care" has been selected by means of an exhaustive search from a dictionary of 43'000 most used words to increase this probability. There are a lot of similar words to "care", so in theory, if every solution is also accepted with a Levenshtein distance of 1, the trivial solving rate is much higher.

For example, all of following 40 words have a Levenshtein distance of 0 or 1 to "care":

care  $\rightarrow$  care are bare cade cadre cafe cage cake came cane cape car card care cared cares carey carl carp carr cars cart carte carve cary case cate cave clare core cure dare fare gare hare mare pare rare scare ware

Most of these word are very common, but nonetheless we can only assume that reCAPTCHAs dictionary will contain a majority of them. But if any of this words is used as a challenge, then "care" is also a solution to it. Suppose only 30 of them are in reCAPTCHAs dictionary of 100'000 words, then the trivial solving rate would be estimated as  $\frac{1}{100000} \cdot 30 = 0.03\%$ .

#### 4.5 Fourth generation of CAPTCHAs

It seems that a few days before publishing this bachelor thesis, the CAPTCHAs changed yet again into what could be the next and fourth generation. Firgure 8 shows some examples. This version shares similarities with the second generation CAPTCHAs, as it has no additional distortions in the images. However, the way the CAPTCHAs are deformed changed and appears to be more strong. I spotted it before and classified it as minor reCAPTCHA generation, see Figure 3(d). Based on a small set that I collected, I can conclude that the variety of the CAPTCHAs changed and most notably also upper case letters appear more often at the beginning of verification words. Additionally I observed names or places more frequently, so that the underlying vocabulary could have changed, too. This makes sense, because upper case letters are used for names and places. By sorting out all upper case words in the older generations, reCAPTCHA effectively reduced the number of names and places that could be verification words.



## under magnetics

### household encrypted

(a) 42 Katheryn

(b) 42 magnetics

(c) 42 encrypted

Lausanne that

ELINEK Solow

(d) Lausanne 42

(e) 42 Solow

Figure 8: Some examples for the assumed fourth major generation of reCAPTCHAs alongside with the solution that was used to solve the CAPTCHA and that got verified by the reCAPTCHA server. All words tagged with the bogus token "42" are read words.

#### **5** Preprocessing

For the third major CAPTCHA generation of reCAPTCHA it makes sense to preprocess the image and filter the additional shape that is placed into the image to distract recognizers. This can be formulated as an AI-Problem on its own: group the information in the image into parts belonging to the word and parts belonging to the additional distortion.

#### 5.1 General Idea

I propose to filter the edge pixel produced by a Canny Edge Detector [12]. A drawback with this approach is that for the recognition of the word after the filtering, we require an algorithm which only relies on the edges of the characters. A holistic word recognition technique, is later proposed for this (see Section 6) and fulfills this requirement.

#### 5.1.1 Problem formulation

The additional edges produced by the elliptic distortions interfere with recognition algorithms. The problem of filtering the additional edges can be reduced to a two-class decision problem on edge pixel, so that we can apply supervised learning algorithms known from machine learning to filter the edges.

There are two classes,  $C = \{E, W\}$ . For any given edge pixel p, decide if p is either an edge pixel of the distorted ellipse (E) or an edge pixel from the word (W).

#### 5.2 Prerequisites

In order to train a supervised classifier, we first have to label a training set. This will be used to train the classifier and a part of the data will be used to estimate its performance. Then we need a set of features, a way of representing the information that should be used to train and predict the class of any given pixel.

#### 5.2.1 Training set

I cleaned 170 CAPTCHAs that had been run through the Canny edge detector as far as possible by hand. The cleaning process consists of pixel deletion only, no pixel were added. This is a tedious process that needs roughly 2-5 minutes per image after some training. By comparing the cleaned and the original image, the edge pixel from the the original image can be labeled as ellipse (E) for all pixel that are different in those two images and as a pixel from the word (W) for all pixel that remained the same.

Figure 9 shows examples of this hand made training set. It is worth noting, that the labels from this training set do not constitute the ground truth, rather a classification made by a human that already has small mistakes.

#### 5.2.2 Estimation of the ellipse center

To construct good features for the classifier, the relation of a pixel to some approximated position of the ellipse is helpful. I thus propose a simple algorithm next, that approximates the center of the ellipse reliably if the black ellipse is the dominant component of the original image. According to my experience this is the case in about 99% of the images.

The approximation of the position of the ellipse is derived with the help of the basic morphological transformations erode and dilate. Intuitively, a dilation shrinks black areas and an erosion extends black areas in all directions on its edges. These transformations are carried out using a kernel, as shown in Figure 10. A kernel can be any arbitrary pixel window of an image. The usual choice is a  $3 \times 3$  pixel mask. It has an anchor point that represents the origin pixel for which the kernel should be computed for.

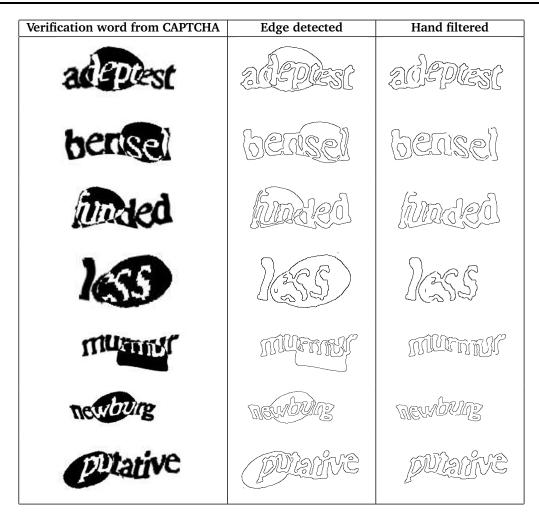


Figure 9: Some examples of the hand made training set.

White regions are stored using higher values in greyscale images than black regions. A dilation in a greyscale image is a function that assigns each pixel the local maximum brightness value over the kernel, while an erosion assigns the local minimum brightness value to the anchor.

$\overleftrightarrow$	

Figure 10: A common  $3 \times 3$  kernel for a morphological transformation. For dilate the pixel with the star is assigned the maximum pixel value over the kernel, for an erosion it is the minimum.

Algorithm 1 is useful to estimate the center of the distorted ellipse in one isolated word. The dilate and erode functions are used with the usual  $3 \times 3$  kernel.  $dilate_n(x)$  means that n successive dilate operations are performed on the image, and  $erode_n(x)$  means n successive erode operations. A vertical projection is the sum of all pixel in a row at each y position, while a horizontal projection is the sum at each x position. The effects of Algorithm 1 on an example CAPTCHA can be seen in Figure 11.

Figure 12 shows an example verification word where the estimation fails. This is because after the erosion, the characters 't' and 's' become the dominant part of the image instead of the ellipse. I counted the occurrence of this problem on a set of 200 CAPTCHAs collected successively on a single day in May 2010. Of the 200 samples, only 2 showed this problem. Hence, assuming that only about 1% of the CAPTCHAs are affected by this problem, no additional steps must be taken into account for this sort of CAPTCHAs.

Algorithm 1: An easy algorithm for approximating the center of the ellipse in the CAPTCHA

Input: Single word from a third generation reCAPTCHA as image

1 repeat

2 CAPTCHA  $\leftarrow$  erode(CAPTCHA)

3 **until** CAPTCHA has a closed shape that is completely filled. See Figure 11(b).

- 4 needed dilates ← number of dilates until CAPTCHA is completely white
- 5 CAPTCHA  $\leftarrow$  dilate<sub>needed dilates-2</sub>(CAPTCHA)
- 6 CAPTCHA  $\leftarrow$  threshold(CAPTCHA)
- $7 x \leftarrow min(horizontal projection(CAPTCHA))$
- 8 y ← min(vertical\_projection(CAPTCHA))

Output: x, y : Estimated coordinates of the ellipse center



"adeptest" as an example

erode operator. The ellipse is the dominant part of image.

(a) The verification word (b) After 7 iterations of the (c) After 15 iterations of the dilate operator on the eroded image. The ellipse is still the dominant of the word.



(d) After 31 iterations of the (e) After 61 iterations in total, (f) After thresholding dilate operator. The rest of the word is now gone at this point.

the image would be just white. This is a snapshot after 58 iterations.



the previous snapshot (colored red) together with the original image. The previous snapshot can be used to easily derive an estimation for the ellipse center.

Figure 11: Example images for the estimation of the ellipse.

#### 5.2.3 Features

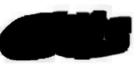
For any given pixel, a vector of features is created. The approximated center of the ellipse from the last section is used for this, the original image, the edge image and the tangents on the edges; that is, for each edge pixel p the direction of the edge at *p*. Each feature is a function  $f_i(q,...)$  with a point *q* and some additional input. *q* represents the position of a given pixel p, for which  $f_i$  should be computed for. The feature vector for p is then  $(x_1, x_2, \ldots, x_n)$  where  $x_i = f_i(q, \ldots)$ .

Table 3 shows an overview and a description of the functions to generate the features that I have used. It is also listed which additional input, besides the position of the pixel, they need.

#### 5.3 Classifier

A classifier can now use the features from the last section to predict the class  $c \in \{E, W\}$  of a pixel. A classifier can be seen as a function h(x), where x is the vector of features  $(x_1, x_2, \dots, x_n)$ , that outputs a predicted class. For the filter problem this is either the class of edge pixel belonging to an ellipse (E) or to a word (W).





(a) Input image

(b) 10x erosion

(c) 45x dilate



(d) After thresholding (colored red) + input image

Figure 12: This is an example there the estimation fails, because after 10 erosions, the characters "ts" become the dominant part of the image. This is very rare and occurred only in about 1% of the verification words from the CAPTCHAs.

Function	Additional input	Description
center_distance_feature	approximated ellipse center	The euclidean distance from $q$ to the approximated center.
angle_feature	approximated ellipse center	The relative angle for $q$ to the approximated center, as if it were the origin in a polar space.
tangent_feature	tangents on the edges	The direction of the tangent on the edge for $q$ . The angle of tangents on all edge points is usually a by-product of the edge computation, for example with the Canny edge [12] detector.
linecount_xaxis_left_canny	edge image	Start at $(x - 1, y)$ from the given point and count the occurrences of other edge pixel on an imagi- nary strait line to $(0, y)$ .
linecount_xaxis_right_canny	edge image	Start at $(x + 1, y)$ from the given point and count the occurrences of other edge pixel on an imagi- nary strait line to $(max_x, y)$ .
linesum_img_feature1	approximated ellipse center, original image	Draws an imaginary line from the approximated ellipse center to $q$ . Then all color values from the original image for all pixel on this line are added together and normalized by the length of the line (in pixel).
linesum_img_feature2	approximated ellipse center, edge image	Same as linesum_img_feature1, but on the edge image.
linesum_img_avg_feature	approximated ellipse center, original image	Same as linesum_img_feature1, but averages the same measurement for all pixel in close proximity, using a $7 \times 7$ mask.
line_nextcannypoint_feature	approximated ellipse center, edge image	The distance on the imaginary line (to the estimated ellipse center) from $q$ to the to the next edge pixel.
avg_color_feature1	original image	The average color in close proximity, using a $7 \times 7$ mask.
avg_color_feature2	edge image	Same as avg_color_feature1, but for the edge image. It can also be interpreted as the density of edge pixel in close proximity.
avg_tangent_feature	tangents on the edges, edge image	Same as tangent_feature, but averages for all edge points in close proximity using a $7 \times 7$ mask.

Table 3: Overview and a description of functions to generate features that can be used for the classification problem.

The distribution of negative and positive examples is in our case biased. In the edge image there are many more pixel that are part of the edge of a character than there are pixel that are part of the ellipse: I could observe a ratio of approximately 1:4. I have chosen AdaBoost [22] as a classifier, because it is known to perform better in those situations than simpler classifiers [56].

"Boosting" refers to a general method to increase the confidence in predictions in computer learning theory. A set of weak classifiers, which can have a performance just slightly above random guessing are used to build a significantly stronger classifier [23].

Since it is in general not known which of the classifiers predicts a sample correctly, AdaBoost uses weights in its learning process. Every weak classifier  $h_i(x)$  is assigned a weight  $\alpha_i$ , that shows how important the classifier is. Then, the value  $\alpha_i$  for each weak classifier is successive learned on the training data, so that the error rate of the end result is minimized. For a more in-depth explanation of the trainings-algorithm I refer to the original papers [22,23] at this point.

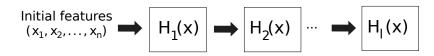
Once the training algorithm is finished, the strong classifier is described as a new function H(x) [10] by the sum over the *n* weighted weak classifiers  $h_i(x)$ :

$$H(x) = sign(\sum_{i=1}^{n} \alpha_i h_i(x))$$

#### 5.4 Classifier cascade

If only one classifier is used, a narrow view is created that decides only depending on the local features of one pixel. This discards completely that an ellipse has some properties, that can be used on a more global view. It would be good to give the classifier some sense of the classifications of surrounding pixel, because the ellipse in the edge image is a connected component. We can also fit an ellipse on the as E classified pixel and exploit the geometric properties of it as a new fitness function to build a new feature.

This is done by a filter cascade  $(H_1(x), H_2(x), \dots, H_l(x))$ : In a first iteration, the classifier  $H_1(x)$  makes its best guest depending on the local features  $(x_1, x_2, \dots, x_n)$  of each pixel. Then, that information is used to compute new features for the next classifier  $H_2(x)$ . Also all initial features that  $H_1(x)$  used are recomputed, because now a better guess for the center of the ellipse can be made and many features rely on this estimation. We do this recursively, so that *i*-th iteration uses the result from  $H_{i-1}(x)$  and computes a new feature vector  $(x'_1, x'_2, \dots, x'_n, x_{n+1}, \dots, x_{n+m})$  with *m* additional features for  $H_i(x)$  to gradually build a better classification based on the previous one.





For each iteration we have to train a new classifier and when filtering and predicting a new image, we must use the cascade in the same way as in the training phase. Figure 13 illustrates the classifier cascade.

#### 5.4.1 Additional features

I used a  $3 \times 3$  kernel and a  $7 \times 7$  kernel over which the average density of pixel previously classified as ellipse was measured. An ellipse fitting function was also computed over all as ellipse classified pixel (see Figure 14(a)). I also used an approximated distance function d(E, p), that gives an estimate about the minimal distance from point p to the edge of the ellipse E as a new derived feature. Figure 14(b) illustrates d(E, p). The approximation was done by transforming the problem to the unit circle, in which the shortest distance problem is easy to solve and re-transforming it to the ellipse. My approach is similar to the approximation proposed in [60].

This helps to give the classifier some notion of the geometric properties of the elliptic object that should be filtered on a larger scale, even though the additional ellipse in the CAPTCHAs is sometimes severely deformed. The data from the previous iteration can be used to derive a better estimate for the center of the ellipse as well as the one Algorithm 1 provides initially.

After the cascade has classified the image, it can be filtered according to the predicted class of each pixel. We can simply remove all pixel that have been classified as an ellipse pixel.



 (a) An ellipse fitting algorithm on the as ellipse classified pixel from the classifier.



(b) The distance function d(E, p). Brighter pixel have a small distance and darker pixel are further away, near zero distance is illustrated with a black color.

Figure 14: Ellipse fitting on the as ellipse classified pixel from the previous classifier. The distance function d(E, p) to the fitted ellipse E can be used as additional feature for the next classifier in the cascade.

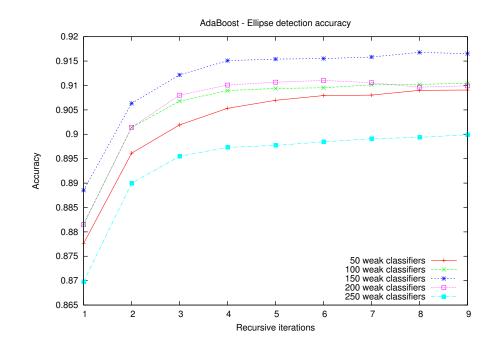
#### 5.5 Results

I used the implementation of AdaBoost and its variants in the OpenCV library, to implement the filter as explained in the last section. The book "Learning OpenCV" [10] by G. Bradski and A.Kaehler gives also an introduction on Boosting algorithms and gives instructions how to use the implementation in OpenCV correctly. The implementation allows to set priors, the cost of misclassifying a class. This is useful to alter the amount of false positives and false negatives. For all results shown here, the prior for misclassifying a pixel as ellipse was chosen as two times higher than the cost for misclassifying a pixel as word. This makes sense, because the pixel that are classified as ellipse get filtered and we do not want to misclassify too many pixel as ellipse otherwise we could risk to loose too much pixel from the word. The classifier cascade with initial and additional features have been implemented as described in the previous sections.

For each run, the hand-labeled training data has been randomly separated into a training and a validation set. I used the training set to train the classifiers with 90% of the images, for estimating their performance on unseen data I used the remaining 10% for the validation set. I varied the amount of weak classifiers and averaged all measurements over 4 runs, in which the classifiers have been individually trained each time. Figure 15 shows the percentage of correctly classified pixel, called accuracy, relative to the hand-labeled data. The measured accuracy is an estimate for unseen data, because the validation set was not used in the training phase for the classifiers. It can be seen that choosing more than about 150 weak classifiers has a negative effect on unseen data, this is most likely a result of overfitting to the training set. As the number of weak classifier should rather learn to generalize from the training set than memorize it.

Figure 16 shows timings for both learning and prediction, measured on a 3.0 GHz Quad-Core computer. Learning takes a considerable amount of time, but is a one time computation. After that, classifying one image is very fast and takes between 100 and 400 milliseconds per image depending on the number of weak classifiers used for 9 cascaded classifiers. Both learning and prediction timings are linear to the number of weak classifiers used.

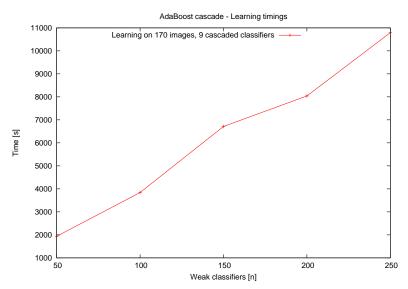
Table 4 shows individual misclassifications in a  $2 \times 2$  confusion matrix for a classifier cascade with 150 weak classifiers. It can be seen that false positive and false negative classifications are quite balanced. Table 17 is meant to give an impression of the quality of results. While some characters have been "damaged" and some parts of the ellipse are not detected, the overall results are quit good considering that the classifiers work on a per pixel basis. The filter works better, if the ellipse does not go through a substantial part of the word. Figure 18 shows an example run of the classifier cascade, with an image of the classified data for each iteration. It can be seen that the final result is iteratively improved.



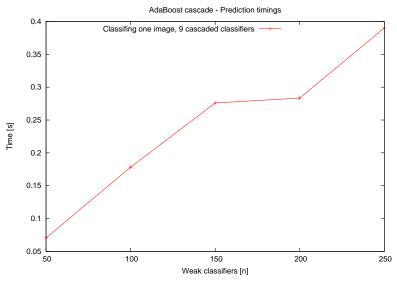
**Figure 15:** Averaged accuracy curve over the iterations, with a cascade of 9 trained AdaBoost classifiers in total. The highest difference can be observed from iteration 1 to 2. Choosing the amount of weak classifiers to high has a negative effect on the accuracy on unseen data.

is	Classified as E	Classified as W	Sum
E in training set	6121.25 (15.6%)	1689.25 (4.3%)	7810.5 (19.9%)
W in training set	1627.75 (4.1%)	29885.25 (76%)	31513 (80.1%)
Sum	7749 (19.7%)	31574.5 (80.3%)	39323.5 (100%)

**Table 4:** The  $2 \times 2$  confusion matrix for the complete classifier cascade with 150 weak classifiers on unseen data. Each pixelis one classification. The amount of pixel in the validation set varies because it varies in each individual image.All numbers shown here are averaged across 4 runs and 4 different validation sets randomly chosen from the<br/>training set.



(a) Time it takes to train 9 cascaded classifiers for different amounts of weak classifiers.



(b) Time it takes to use 9 cascaded classifiers for predicting one image.

Figure 16: Timings for learning the cascade of classifiers and using it for prediction. Learning takes a considerable amount of time, after that prediction is very fast.

Verification word	Edge image	Classified as word	Classified as ellipse
auctions	avcilions	ALY CILLOUNS	
Cosiest	COSIESI	COSIES	ì
compete	COMPECE	COMPEC	rr
मान्य	PULDICS	PMDits	
snippets	SMIPPELS	STAN FPECS	
nonfatal	(Indialization)	INC MARTAN	
salaams	salaents)	selenns,	Ē
witindrew	WENDERRY	WIDIEW	
wounded	CryOnnal Est	WOUNDELED	

**Figure 17:** Some examples for classifications on new data, to give an impression of the quality of the classification. Noticeable is that the classification works better when the severeness of the additional distorion is only a "slight deformation", as opposed to a stronger deformation (see also Table 2).

Iteration	Filtered CAPTCHA	As ellipse classified pixel
0	(DINDAC) EO	
1	(DINDODEO)	
2	BANDODEO	
3	DANDEDED	
4	DAINE	
5	DAUDODED	
6	DUIVIONED	
7	DAVIDODEO	
8	DAUDODEO	
9	[DITIVITO]] & D	

**Figure 18:** An example prediction of the classifier cascade. This verification word did not belong to the trainings set, the distorted ellipse goes through parts of three characters. Some pixel have been falsely classified in the first iteration, but the classification is good enough to help the next classifiers in the cascade to gradually build a better classification.

#### 6 Holistic word recognition

#### 6.1 Motivation

In the analysis section, it has been outlined that reCAPTCHA is an implicit segmentation resistant CAPTCHA and uses mostly words from the English language, so that the space of possible solutions is limited in respect to that.

Let us assume that reCAPTCHA is truly segmentation resistant and that there is no way to segment the words from the CAPTCHA into smaller entities than the word itself. Then, with a recognition on word basis, we could still solve the CAPTCHA without breaking the assumption that reCAPTCHA is segmentation resistant. The elegance of using word recognition for a segmentation resistant text CAPTCHA, is that we do not need to find holes in the assumption or need to prove that there is an algorithm that can segment the CAPTCHAs; we simply try to solve the CAPTCHAs and ignore the segmentation issue.

A holistic word recognizer needs to consider the whole search space of all possible words, like a character recognizer would need to consider the (small) space of all characters. The problem gets computationally more difficult as the search space gets bigger. However, for a search space in the magnitude of 40'000 to 100'000 possible solutions (the approximated size of all unique reCAPTCHA solutions as of today, see discussion in Section 4.3.4) I show that a holistic word recognizer using object recognition is feasible.

Finally, a holistic recognizer for a word CAPTCHA is also inspired by psychological studies. If recognition on word basis helps humans to decipher words, the same process will possibly help computers. After all, if we can mimic the way we humans read computationally, we will solve any word CAPTCHA, because they have to be easy to read for humans.

#### 6.2 Shape contexts

Shape contexts can be used to build a holistic word recognizer. The external boundary of an object is called the shape of an object. Mori et al. defined the notion of a *shape context* [43] for such a shape, a mathematical construct that is useful to compute the similarity between two shapes. This can be used to measure the similarity between two objects, by computing the external boundaries first and then defining a distance measure on the boundaries. Finally, we can search the most similar object in a space of known objects, by finding the object with the highest similarity given by the smallest distance.

By defining words as objects, we can construct a database of boundaries of known words. Then, to recognize a new word, we can compute the similarity to each word in the database, using the most similar word as a result to the search. This can be compared to searching a word in a dictionary, but doing so with just the visual representation without any literal semantics. This is a holistic recognizer - words are not segmented into all their individual letters. In cases where segmentation into letters is difficult, this in an advantage. The drawback besides computational costs, is that we are only able to recognize words that are part of the database. Using shape contexts for comparing the words has the additional advantageous property, that is still possible to recognize words if they are slightly deformed. Belongie et al. [8] were able to show experimentally that shape contexts are invariant under scaling and translation and robust under small geometrical distortions, occlusion and presence of outlier.

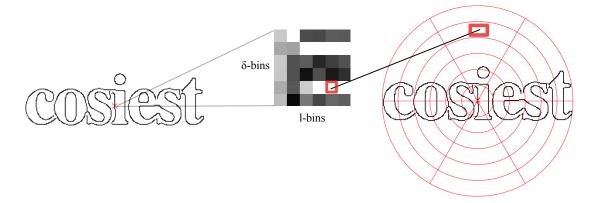
#### 6.2.1 Building a shape context

For the external boundary of the word, we use the output of a Canny edge detector. We can also filter the edges, as discussed in the section "Preprocessing". If we look at the edges as a set of discrete pixel, each of them is a point in  $\mathbb{N}^2$ . Let *P* be such a set, with |P| = n points. A shape context defines a relation between a point  $p \in P$  to all other n - 1 points in P. The point *p*, for which the relation is built, shall be called *master point*. Defining the master point as the origin, we build n - 1 vectors to the other points. Then, we transform them to the polar space, so that each vector is defined as the relative distance and relative angle to the master point. The shape context for a master point *p*, is a two dimensional histogram that has a chosen number of radial bins and length bins. Radial steps and length steps divide the length and angles in bins of equal length <sup>3</sup>. The number of vectors that point to each of the bins is counted, building the two dimensional histogram. Figure 19 illustrates this. For each shape, we generate many such histograms, each from a different master point.

Then, the general idea is to build point correspondences between two shapes, based on the similarity of the respective histograms, by using a distance function to compare the histograms. One major goal is to be able to define a distance function for two individual shapes, that can also be used to find the most similar shape among a set of other shapes by searching the best matching pair.

It is useful to normalize the histograms before comparing them, for example by normalizing the standard deviation ( $\sigma$ ) and expected value ( $\mu$ ) of each individual value of the histogram:  $H_{norm}(i, j) = \frac{H(i, j) - \mu_H}{\sigma_H}$ .

<sup>&</sup>lt;sup>3</sup> In the shape context proposal by Mori et al. [43], the length steps are logarithmic. In my experiments, I have used equal lengths as they turned out to be slightly better for word matching.



**Figure 19:** Center of the figure: A shape context histogram from a master point in the center of the word "cosiest" visualized. The particular master point is marked on the left. All other edge points are relative to the master point. For the bin sizes, 6 length and radial steps of equal length have been chosen. To visualize the bins, their borders have been transformed back to the Cartesian space to the word on the right. One of the bins is marked in the histogram and in the word on the right, to give an orientation. The histogram counts how many edge points belong to a certain bin. White colors in the histogram indicate no pixel in the particular bin and darker colors indicate many pixel in the bin.

#### 6.3 Discrimination of shapes

Because the list of potential words is large, a method for rapid coarse discrimination of shapes is needed. I propose an approach similar to the fast pruning method in the shape matching framework by Mori and Belongie et al. [8, 43]. An appropriate distance function is presented in the next sections. Small modifications are made, to customize for the word recognition of reCAPTCHA words. Then, in Section 6.5 a probabilistic search algorithm is proposed, that is useful to find a good match in vast search spaces (for example  $10^4 - 10^5$  shapes) and is practical on todays home computers.

#### 6.3.1 Comparison

We can interpret a two dimensional histogram as a one dimensional one, by looking at the bins in a sequential order. Two such histograms can now be compared by the following  $\chi^2$  distance function  $D_1$ , where we compare two histograms of the same length *l*:

$$D_1(H_1, H_2) = \frac{1}{2} \cdot \sum_{i=0}^{l} \frac{(H_1^{\ i} - H_2^{\ i})^2}{|H_1^{\ i}| + |H_2^{\ i}|}$$

The result is always positive and a small distance  $D_1(H_1, H_2)$  signals that the histograms  $H_1$  and  $H_2$  have a high similarity.

Each shape *s* has a set of master points  $P_s = (p_s^{1}, ..., p_s^{l})$  associated with their respective set of histograms  $M_s = (H_s^{1}, ..., H_s^{l_s})$  so that the *i*-th histogram to  $p_s^{i}$  is  $H_s^{i}$ . Let  $M_1$  and  $M_2$  be two such sets for two shapes  $S_1$  and  $S_2$  that should be compared, both of the same length  $l_1$  and with  $M_1^{i}$  and  $M_2^{i}$  being the *i*-th histogram of those sets. To correlate them we must combine the histograms, so that the sum over the previous defined distance for histogram tuples of  $M_1$  and  $M_2$  is minimized:

$$D_M(M_1, M_2) = \min_j \left( \sum_{i=0}^{l_1} D_1(M_1^{i}, M_2^{j}) \right).$$

Belongie et al. proposed for the detailed matching part [8] in the matching framework, to form a bijective association between the histograms of  $M_1$  and  $M_2$  and minimize the above sum in respect to that property, in other words this guarantees that every  $M_2^{j}$  is chosen exactly once. The result is then a point correspondence between all master points of  $P_1$  and  $P_2$ . For this, however, a  $n \times n$  distance matrix between all histograms must be calculated and bipartite matching algorithms can be used to find the best assignment, so that the sum of corresponding histograms is minimized. The problem to find such a minimum is called the *assignment problem* and it can be solved by a polynomial time algorithm called the Hungarian method [35]. The original Hungarian method performs in  $O(n^4)$  and with further refinements in  $O(n^3)$  [31].

#### 6.3.2 Appearance similarity

To save computational costs, we can refrain from a strict bijective association and choose j freely in  $D_M$ . Then, if we just search the smallest  $D_1$  for every  $M_1^{i}$  from  $M_1$  we can end up with some  $M_2^{j}$  from  $M_2$  being used more than once and other  $M_2^{j}$  would be unused. As long as the two histograms do not belong to totally different parts of the words, this is not critical. The idea is, that the master points should be somehow on similar positions. For this, we can use a different constrain: We can define a distance for the local appearance similarity of two master points. We use the observation from the reCAPTCHA words, that distortions have much more influence on the y-axis than on the x-axis. We can restrict the search to master points at similar places on the x-axis and can prefer master points that are closer together on the x-axis. Thus, a simple  $D_{local}^{-1}$  is:

$$D_{local}{}^{1}(p_1, p_2) = \left| relative_x(p_1) - relative_x(p_2) \right|$$

where  $p_1$  and  $p_2$  are two master points and  $relative_x(p)$  returns a x-position value relative to the shape width from the shape of point p.  $D_{local}^{-1}$  should be normalized so that it returns a value between 0 and 1.

This can then be used to restrict D:

$$D((M_1, P_1), (M_2, P_2)) = \min_{j} \left( \sum_{i=0}^{l_1} D_1(M_1^{i}, M_2^{j}), \text{ where } D_{local}^{-1}(P_1^{i}, P_2^{j}) < t_x \right).$$

where  $P_m^i$  is the respective master point to the histogram  $M_m^i$  and  $t_x$  is some threshold for the relative x position. The smaller  $t_x$  is chosen the more histogram comparisons for  $D_1$  can be skipped, but if it is too small, we may loose the ability to find the best master point combinations for deformed shapes, which would have master points that are further away, which would not desirable for the reCAPTCHA words.

A second local distance can be built for the direction of the edge in the master point p. The idea is, that the local similarity is also dependent on the edge p belongs to and we can decide to skip a comparison between two histograms, if the respective master points are on entirely different oriented edges. An appropriate measurement is the tangent angle dissimilarity [8]. Thus,  $D_{local}^2$  is:

$$D_{local}^{2}(\theta_{1},\theta_{2}) = \frac{1}{2} \left\| \begin{pmatrix} \cos(\theta_{1}) \\ \sin(\theta_{1}) \end{pmatrix} - \begin{pmatrix} \cos(\theta_{2}) \\ \sin(\theta_{2}) \end{pmatrix} \right\|$$

where  $\theta_1$  and  $\theta_2$  is the respective edge orientation in some master points  $p_1$  and  $p_2$ . The values for  $D_{local}^2$  range between 0 and 1, where smaller values signal a more similar edge orientation.

Analogue to  $D_{local}^{1}$ , D can be extended by  $\Theta_{1} = (\theta_{1}^{1}, \dots, \theta_{1}^{l})$  and  $\Theta_{2} = (\theta_{2}^{1}, \dots, \theta_{2}^{l})$  so that  $\theta_{x}^{i}$  is the respective edge direction in a master point  $p_{x}^{i} \in P_{x}$ . Then, with the new distance  $D_{local}^{2}$  we have:

$$D((M_1, P_1, \Theta_1), (M_2, P_2, \Theta_2)) = \min_j \left( \sum_{i=0}^{l_1} D_1(M_1^i, M_2^j), \text{ where } D_{local}^{-1}(P_1^i, P_2^j) < t_x \wedge D_{local}^{-2}(\Theta_1^i, \Theta_2^j) < t_\theta \right).$$

where again  $t_{\theta}$  can be additionally used as a threshold to restrict the search for the minimum to a smaller set of histograms.

#### 6.3.3 Generalized shape context

Mori et al. [44] proposed an extension to the shape context descriptor, called generalized shape context. The idea is to create a richer descriptor, by using more information from the source image to create the histograms. They proposed to use the direction of the edges as additional information. For each bin of the normal shape context the average direction of all tangents from all edge points is computed. I used this idea to represent the edge information in an additional histogram,  $H_{\Theta}$  with the individual averaged direction  $H_{\Theta s}^{i}$  in *i*-th bin normalized to  $\pm 1.0$ . Let  $M_{\Theta s}$  be a set of such additional histograms  $(H_{\Theta}^{1}, \ldots, H_{\Theta s}^{l})$  then  $D_{1}$  can also be used as  $D_{1}^{\Theta}$  to compare those two histograms.

With the help of the generalized shape context and the local appearance of the master points a better distance function can be built. We can add all individual distances together and assign them weights, that can be used to fine tune  $D_s$ :

$$\begin{split} D_{S}((M_{1}, M_{\Theta_{1}}, P_{1}, \Theta_{1}), (M_{2}, M_{\Theta_{2}}, P_{2}, \Theta_{2})) &= \\ \min_{j} \left( \sum_{i=0}^{l_{1}} \left( \alpha D_{1}(M_{1}^{i}, M_{2}^{j}) + \beta D_{1}^{\Theta}(M_{\Theta_{1}}^{i}, M_{\Theta_{2}}^{j}) + \gamma D_{local}^{1}(P_{1}^{i}, P_{2}^{j}) + \delta D_{local}^{2}(\Theta_{1}^{i}, \Theta_{2}^{j}) \right), \\ \text{where } D_{local}^{1}(P_{1}^{i}, P_{2}^{j}) < t_{x} \wedge D_{local}^{2}(\Theta_{1}^{i}, \Theta_{2}^{j}) < t_{\theta} \right). \end{split}$$

We can use  $D_S$  now to compare two shapes  $S_1$  and  $S_2$ , by generating the respective shape contexts and master points  $(M_i, M_{\Theta i}, P_i, \Theta_i)$ . Then,  $l_1$  is the number of shape contexts in the first shape and j can be chosen freely so that  $j \leq l_2$ , where  $l_2$  is the amount of shape contexts of the second shape, as long as the constraints for the local distance functions with the thresholds  $t_x$  and  $t_{\theta}$  are met.

Algorithmically,  $D_S$  is simple to implement: For every shape context of the first shape, we first reduce the search space in the second shape by the local distance functions, in other words find possible histogram candidates of the second shape for a closer search based on pruning with the appearance similarity. Under all candidates we extensively search the one that would yield the smallest combined distance  $\alpha \cdot D_1 + \beta \cdot D_1^{\Theta} + \gamma \cdot D_{local}^{-1} + \delta \cdot D_{local}^{-2}$  (all parameters omitted for clarity). If there is no candidate left after pruning with the local distance functions, we can assign a penalty value  $\varepsilon$ instead of the smallest combined distance.

The two shapes do not need to have the same amount of master points and associated shape contexts, but because we iterate over the length  $l_1$ , it is favorable if the amount of shape contexts differs to use the shape with the smaller amount as the first parameter in  $D_S$ .

 $D_S$  can be computed in  $O(l_1 \cdot l_2)$  with some constant factor  $c_d$  for calculating the combined distance function and a constant pruning factor estimated by  $t_x \cdot t_{\theta}$ , with the very naive assumption that the two local distance functions return values uniformly distributed from 0 to 1.

#### 6.4 Database

There are two possibilities to generate the database of shapes, that is used to find the best match for a word CAPTCHA: We can collect the data from the CAPTCHA server and label it, or we can generate our own shapes. The generator for the CAPTCHAs of reCAPTCHA is not publicly available, and the CAPTCHAs change significantly from time to time. It also costs too much manual labor to collect and label a significant amount of samples.

Thus, it makes sense to implement a custom generator, that simply renders words from a font without distorting them. We can, however, make use of three observations that are valid for all major CAPTCHAs up to this date: The majority of verification word is in fonts with serifs and characters do touch each other or overlap very often, so that the word is segmentation resistant. Also the majority of the words can be found in an English dictionary. To mimic this situation, we generate our words from a list of most used words in the English language. For this we use a serif font and set its letter-space parameter to a negative value, so that the characters overlap a bit. When we compute the shapes with the canny edge detector [12] as edges of the generated words and calculate the shape contexts from it.

If we use our own generator for the shapes of words, we could even compute our database adhoc while running our search for the most similar match. It still makes sense to precompute the database, to save computation time. But there is no real learning phase involved: The generated shapes can be seen as static and ideal templates to which we compare the real CAPTCHA words to. (Only the search for suitable parameters could be interpreted as a learning/adapting phase.) But unlike with a classifier that learns from labeled and real data, by using generated templates we are quite flexible if the words that we want to recognize change significantly; we adjust the parameters for the generator accordingly if needed. There is not need to recollect labeled sample data. Thus, the resulting word recognizer is quite generic.

#### 6.4.1 Selecting master points

If we build the database of shapes naively, we take each edge point of the word and use it as a master point. The amount of master points will vary from image to image, furthermore storing and generating shape contexts for all points could produce too much data for a large number of words. As a countermeasure, we could produce smaller images and thus use less edge points. However, some words with more points would still be overrepresented in the database. We can choose k master points at random from each word, where k is a constant number. Then each word has the same number of shape contexts. Figure 20(a) illustrates this with 16 points. We can still use all other points to build the shape context for each of the master points, so that they are more accurate.

To guarantee a more evenly spaced selection, we can use a cluster algorithm to build k clusters from all points with the euclidean distance. The centroids of those clusters will be evenly spaced apart. However, the well known K-means cluster algorithm [39] iteratively builds better clusters around averaged centroids, that are not part of the data; in our case we would have master points that are not edge points any more. There is a variant of K-means called K-Medoids

that uses existing points from clusters as centroids, so that instead of averaging the points in a cluster in each iteration, it searches the best point from the cluster as centroid that minimizes the distance to all other points in a cluster. K-Means and K-Medoids converge iteratively to a local minimum solution (to the cluster problem) and the algorithms terminate if no better clusters can be found in a new iteration. Figure 20(b) shows the effect of using K-Medoids to select 16 master points. <sup>4</sup>



 (a) 16 master points selected uniformly at random



(b) 16 master points selected as centroids from the K-Medoids algorithm

Figure 20: Examples for selecting k=16 master points in different ways on a generated word.

#### 6.5 Search algorithms

In the next sections, I present two different search algorithms that can be used to find the best matching shape in a database of shapes for a query shape.

#### 6.5.1 Naive search algorithm

Algorithm 2 shows a naive search algorithm, that simply compares every set of shape contexts for one word in the database to the input image and outputs the (first) word with the smallest distance on the whole dictionary.

Algorithm 2: NaiveSearch: A naive search algorithm using the distance function  $D_s$ .

**Input**: img: Image of verification word (query shape), db: Database of *n* shape contexts sets  $((M_1, M_{\Theta_1}, P_1, \Theta_1), \dots, (M_n, M_{\Theta_n}, P_n, \Theta_n))$  for *n* words  $(w_1, \dots, w_n)$ 

1  $(M_s, M_{\Theta s}, P_s, \Theta_s) \leftarrow \text{generateShapecontexts(img)}$ 

2 bestDistance  $\leftarrow \infty$ 3 bestWord  $\leftarrow \varepsilon$ 4 foreach  $(M_d, M_{\Theta d}, P_d, \Theta_d)$  in db do 5 | distance  $\leftarrow D_S((M_s, M_{\Theta s}, P_s, \Theta_s), (M_d, M_{\Theta d}, P_d, \Theta_d))$ 6 | if distance < bestDistance then 7 | bestDistance  $\leftarrow$  distance 8 | bestWord  $\leftarrow w_d$ 9 | end

10 end

Output: bestWord: The most similar word in the database

The function *generateShapecontexts(image)* generates a set of shape contexts and master points with respective tangents as described in Section 6.2.1. It must use the same bins sizes, that have been used to generate the database db.

#### 6.5.2 Better-half-search algorithm

The naive search (Algorithm 2) can find a good match, however if the used dictionary for the database is large, the number of comparisons with different master points we need for each shape has to be set very high to compensate for the occurrence of many close matches with similar words. To minimize the computational costs, while still being able to differentiate between small differences, we can use the observation that on the one hand there are many close matches with similar words, but on the other hand also many more very different words that are easy to distinguish by a small amount of shape contexts. To put it differently: it is easier to divide the search space in a group of good matching words and bad matching words by a small amount of comparisons, then trying to find the best matching word directly with that same amount. A similar idea is present in the shape matching framework of Belongie et al. [8], that they called "Fast pruning".

<sup>&</sup>lt;sup>4</sup> k=16 was used for illustrative purposes, actual numbers for k should be chosen bigger.

Algorithm 3: Better-half-search algorithm using the distance function  $D_S$ 

**Input**: img: Image of verification word (query shape), db: (local copy) Database of n shape contexts sets  $((M_1, M_{\Theta_1}, P_1, \Theta_1), \dots, (M_n, M_{\Theta_n}, P_n, \Theta_n))$  for n words  $(w_1, \dots, w_n)$ ,  $n_s$ : start size of sample points,  $f_g$ : growth factor of sample points in each iteration,  $t_f$ : threshold value for the final stage,  $n_f$ : sample points for the final stage.

1  $(M_s, P_s, \Theta_s) \leftarrow \text{generateShapecontexts(img)}$ 2 samplepoints  $\leftarrow n_s$ 3 repeat  $(M'_s, M'_{\Theta_s}, P'_s, \Theta'_s) \leftarrow \text{randomSubset}((M_s, M_{\Theta_s}, P_s, \Theta_s), \text{samplepoints})$ 4 foreach  $(M_d, M_{\Theta d}, P_d, \Theta_d)$  in db do 5 distance<sub>*I*</sub>  $\leftarrow$   $D_S((M'_s, M'_{\Theta s}, P'_s, \Theta'_s), (M_d, M_{\Theta d}, P_d, \Theta_d))$ 6 end 7 sort  $(w_1, \ldots, w_n)$  by (distance<sub>1</sub>, ..., distance<sub>n</sub>) in ascending order 8 **for**  $w_i$  from  $w_{\lceil n/2 \rceil}$  to  $w_n$  **do** 9 10  $delete(w_i)$ end 11  $n \leftarrow \lceil n/2 \rceil$ 12 samplepoints  $\leftarrow [f_g \cdot \text{ samplepoints }]$ 13 14 until  $n < t_f$ 15 bestWord ← NaiveSearch2(randomSubset( $(M_s, M_{\Theta_s}, P_s, \Theta_s), n_f$ ), db) Output: bestWord: The probably most similar word in the database.

I propose a new search algorithm, that I call better-half-search (see Algorithm 3), which extends this idea and divides the search space iteratively in two groups by sorting the distance measures obtained from  $D_S$  and continues the search on the group of better matching shapes with a higher precision. A subset of master points and respective shape contexts – that shall be called sample points – are chosen at each iteration from the first shape at random. As the shapes that are still left in the search space become more similar, the amount of sample points is increased. In each iteration we multiply it by a constant growth factor  $g_f$ . After less than  $t_f$  words are left, we run the naive search with a constant and bigger number  $n_f$  of sample points on the remaining words, that matched well during the iterative process. We can also set  $t_f = 2$  and spare the final naive search.

Algorithm 3 uses the function  $randomSubset((M_s, M_{\Theta_s}, P_s, \Theta_s), samplepoints)$  that returns a random subset  $(M'_s, M_{\Theta_s'}, P'_s, \Theta'_s)$  of all  $(M_i, M_{\Theta_i}, P_i, \Theta_i)$ . NaiveSearch has to be slighty modified into NaiveSearch2 to take the shape contexts and master points  $(M_s, M_{\Theta_s}, P_s, \Theta_s)$  as input instead of an image. The database db is a local copy, so that words can be simply removed from the search space.

The advantage of Algorithm 3 is that we can start with a very small number of sample points  $n_s$  in the first iteration. Let  $D_s$  be our distance function, that can be computed in  $l_1$  constant steps, if we set  $l_2$  to a constant value.  $l_1$  is the number of sample points chosen in line 4. There are n words compared in Algorithm 3, which gives  $n_s \cdot n$  needed constant steps for the first iteration. If we set  $f_g \leq 2$  and  $t_f = 2$ , when the number of total constant steps would be bound by the following progression for  $log_2(n)$  iterations:

$$n_s \cdot n + 2 \cdot n_s \cdot \frac{n}{2} + 4 \cdot n_s \cdot \frac{n}{4} + 8 \cdot n_s \cdot \frac{n}{8} + \ldots = \log_2(n) \cdot n_s \cdot n.$$

If we would just do NaiveSearch2(randomSubset( $(M_s, P_s, \Theta_s), s$ ), db) on the whole db, we would need  $s \cdot n$  constant steps. Thus, the better-half-search gives a runtime improvement over the naive search, if we choose:

$$n_s < \frac{3}{\log_2(n)}$$

The number of sample points grows to  $f_s^i \cdot n_s$  in the *i*-th iteration, which can give us a very accurate discrimination of shapes for more similar shapes through the exponential growth after a few iterations, even for small  $n_s$ . This gives the possibility of choosing very small values for  $n_s$  to gain a significant runtime improvement over the naive search, under the assumption that  $n_s$  start sample points suffice to sort out the  $\frac{n}{2}$  worst matches.

Also worth noting is that the better-half-search is probabilistic, that is, it does not necessarily find the shape with the smallest distance  $D_S$ , as not all shape contexts are used for every comparison. It can still find the best matching shape with some probability and most likely outputs a shape that has also has a small distance  $D_S$  otherwise. This is not so bad because the shape with smallest distance  $D_S$  to the query word is not necessarily the solution to the query. However, if the naive search is used on all available shape contexts, it finds the shape from the database that has the smallest distance  $D_S$  and is at the expense of computational costs non-probabilistic.

#### 6.5.3 Pruning with single characters

The words might be difficult to segment into all individual characters, but the problem of segmentation is considerably easier for the first and last character of a reCAPTCHA word. Unlike with characters from within the word, we immediately know where the first character starts and where the last character ends. For a very naive approach we could try to cut the last or first character of the word with a fixed averaged width for a single character. Then we build a database of all 26 (lower case) characters and try to find the best match for this single character with the naive search algorithm, because the solution space is very small. If the recognition succeeds in a significant number of cases, we can use this information to individually prune the words used in our shape database for each run, so that the word recognizer does only compare all words with the recognized first or last character. This saves a significant amount of computation time and could also improve final matching results, depending on the accuracy of the character recognition.

However, a suitable solution with Levenshtein distance 0 or 1 is most likely pruned from the solution space, if the character recognition fails. Thus, it makes sense to control the amount of character pruning, by extending Algorithm 2 so that it returns a list of  $n_c$  best matching characters for the cut part of the word. Instead of relying on a single best match for pruning, we prune the shape database with best matching  $n_c$  characters. If the solution to the query word is in the database, it remains in the search space after pruning with the probability of having recognized the correct character in the list of  $n_c$  best matches, which should be considerably higher as the probability for the single best match. It should also be worth noting, that a suitable solution with Levenshtein 1 could be pruned away from the search space, because only the last character of this solution is different. If we use a list of  $n_c$  best matches, this solution with Levenshtein distance 1 to the correct solution has at least a chance to remain in the search space.

#### 7 Final results

A holistic reCAPTCHA solver on shape context basis, which I named *Shapecaptcher*, was implemented in C++ as described in the previous sections using the proposed word recognition technique, preprocessing and better-half-search (see Algorithm 3) with pruning on character basis. I tested and experimented first with a set of 1007 verification words obtained in May 2010, collected with the method described in Section 4.1. Then, I tested a set of about 500 verification words from November 2009, which are from second generation reCAPTCHAs. Shortly after the third reCAPTCHA generation changed (end of July 2010), I obtained a set of 300 CAPTCHAs from the new reCAPTCHA generation, to also include a performance estimate for this sort of CAPTCHA. Solving many verification words at once can be trivially parallelized by solving *n* successive verification words in parallel. I measured all timings on a PC with 8GB RAM and an Intel Quad core (Q9550) that I use with a frequency of 3.0 GHz, by using all four cores in parallel with four threads.

The third generation can also be solved at a much smaller rate, without any preprocessing; to demonstrate the positive effect of preprocessing they have been both tested with and without the filter described in Section 5. Table 5 shows the results for the May 2010 set. The table headings have been abbreviated, whereas:

- *S* is the settings profile used, I created two different profiles; "high" is meant for a word recognition with good accuracy and "fast", as the name implies, is tuned for a faster computation (at the cost of accuracy).
- D is the used dictionary, with |D| being the size of the dictionary.
- *l*/θ bins are the two dimensions (length bins and radial bins) for the histogram size chosen to build the database for the experiments. It directly effects the size of the resulting shape database from the dictionary.
- $R_w$  is the total success rate for the whole set and is the fraction of words that are recognized within Levenshtein distance 0 or 1.
- $R_w$  to *D* is the success rate of words under the premise that the verification word is actually in the dictionary with Levenshtein distance 0 or 1, that is, the fraction of words correctly recognized within Levenshtein distance 0 or 1 under all words for which the recognizer has a chance to output a correct solution.
- $R_c$  is the success rate of recognizing the first (f) or last (l) character. If the comparison is set to output multiple characters as hypothesis  $(n_c)$ , than it it the success rate of containing the correct character in the set of multiple hypothesis.
- $n_c$  are the n best matching characters chosen for the first or last character pruning.
- $T_{avg}$  is the average time it takes to output a solution for one verification word.
- $L_{avg}$  is the average Levenshtein distance of the recognition compared to the correct solution of the verification word.
- $P_{avg}$  is the average result of pruning, that is, the number of words left in the search space for the word recognizer, after character pruning has been applied.

S	$1/\theta$ bins	D  <b>in GB</b>	$R_w$ total	$R_w$ to <b>D</b>	R <sub>c</sub>	n <sub>c</sub>	filter	T <sub>avg</sub>	Lavg	Pavg
high	12/6	6.2GB	5.5%	9.57%	65.9%	3	yes	12.13 s	4.85	n/a
high	12/6	6.2GB	5.9%	10.43%	73.2%	5	yes	17.5 s	4.8	n/a
high	6/6	3.2GB	3.9%	6.96%	58.6%	1	yes	6.4 s	5.08	2000
high	6/6	3.2GB	4.9%	8.52%	73.4%	3	yes	13.7 s	4.99	4564.43
fast	6/6	3.2GB	5%	8.87%	69.6%	3	yes	2 s	5.14	4435.8
high	6/6	3.2GB	5.3%	9.2%	80.3%	5	yes	20.23 s	4.90	6738.03
high	6/6	3.2GB	4.4%	7.82%	84%	7	yes	26.37 s	4.99	8702.6
high	6/6	3.2GB	0.9%	1.6%	63.3%	3	no	14.5 s	5.30	4755.17
high	6/6	3.2GB	1.7%	3%	72.2%	5	no	20.58 s	5.13	7158.27

#### Table 5: Final results for 1005 third generation CAPTCHAs obtained in May 2010

As one can see, small recognition rates are possible without any preprocessing and recognition rates are much higher with preprocessing. For the experiments with preprocessing, I used character pruning together with the information from the fitted ellipse from Section 5.4.1, so that the last or first character was chosen for recognition depending on which one is further away from the fitted ellipse. The average Levenshtein distances  $L_{avg}$  from all recognized words to the respective correct word correlates with final recognition results ( $R_w$  total). Pruning with just one solution for the single character recognition reduces the search space to roughly 10% of all words on average. This is a bit higher than  $\frac{1}{26}$  as one could assume, but quite naturally some last or first characters of a word occur more often than others and if we recognize a substantial number of characters correctly, then our recognitions follow a similar distribution. Pruning with multiple solutions for the character recognition  $(n_c)$  gives greater numbers for the average number of word left after pruning  $(P_{avg})$ , but final word recognition rates are higher, even though the search space for the word recognizer is much bigger. But due to this bigger search space, more time is needed on average to solve a verification word.

The following settings for the word recognizer have been used for all experiments with the "high" profile: 64 start sample points for the initial comparison, growing by a factor of 1.5 in each iteration of the better-half-search and a final stage of 500 comparisons after less than 200 words are left in the search space. For the "fast" profile, just 4 sample points are used at the beginning, growing by a factor of 1.5 and after 50 words are left the naive search starts with (still) 500 comparisons. For character recognition a second database was build with the same  $l/\theta$  bins indicated for the word recognition. Because only 26 characters must be compared, NaiveSearch was chosen with 500 comparisons in both profiles and repeated 5 times, after which the results have been added together and a number of best matching characters as indicated with  $n_c$  was chosen for pruning. Only for the third generation CAPTCHAs, the "high" profile did not offer measurable more accuracy than the "fast" profile, but the "high" one is approximately 7 times slower.

For all experiments I used the word list from K. Vertanen with 22'282 words, made from an intersection of 10 different English text corpora <sup>5</sup>. Table 6 shows the coverage of the word list in respect to the different CAPTCHA sets. It took about 43 minutes to build the shape database with maximal 10 iterations of K-Medoids (usually K-Medoids finds a convergence earlier) and 6 bins for the length and radial steps, resulting in shape database of 3.2 GB. It is favorable for computation times if the database can be loaded and stored completely in memory, but that should be no problem with this size for most home computers of today. I used the Pango framework [54] to generate words with the font "Times LT Std Bold" in 40 pt. The font is a serif font and the space between the letters has been set to "-5", so that the letters touch each other similar to the reCAPTHCA word.

CAPTCHA set	Size	L. distance 0	L. distance 0 or 1
November 2009	496 (100%)	149 (30%)	286 (57.7%)
May 2010	1005 (100%)	282 (28.1%)	575 (57.2 %)
July 2010	301 (100%)	59 (19.6%)	149 (49.5%)

Table 6: Coverage of the word list with 22'282 words in respect to the different obtained CAPTCHA sets. Numbers are<br/>indicated for a Levenshtein distance 0, i.e. number of direct hits in the word list, and for a coverage with Leven-<br/>shtein distance 0 or 1. No distinction was made between lower and upper case letters. Nonetheless, coverage<br/>figures there significantly smaller for the new July CAPTCHAs (obtained on July 26, 2010).

The weights for the distance function  $D_s$  and the  $6 \times 6$  database have been set to  $\alpha = 2$ ,  $\beta = 1$ ,  $\gamma = 20$ ,  $\delta = 20$ . The weights  $\gamma$  and  $\delta$  for the local distance functions must be interpreted relative to the area of the histogram bins, because the presented histogram distance function  $D_1$  in Section 6.3.1 is not normalized in respect to bin sizes. The thresholds have been set  $t_x = 0.1$  and  $t_{\theta} = 0.6$  for the word recognition. For the comparison of single characters  $t_{\theta}$  was unused (i.e. set to 1.0) and  $t_x = 0.6$ , because through the naive segmentation of the single character we could compare just a part of a character or more than a single character, thus the thresholds should not be set too strict. Also  $\gamma$  and  $\delta$  have been adjusted for the single character comparison to half the weight used in the word comparison. All these indicated numbers are just recommendations and they have been chosen by means of experimentation. Clearly, an exhaustive search or some optimization technique could yield better values.

As outlined earlier, the better-half-search algorithm is probabilistic (see Section 6.5.2), so that the results have some variance. Earlier experiments showed a maximal deviation of  $\pm 0.35\%$  for  $R_w$  across 10 runs. This is acceptable for estimating the performance, so that the results presented here have not been averaged and are estimates. First experiments where made with a database of 12 length bins and 6 radial bins, but final results where only slightly inferior to a database with half the size and 6 length bins and 6 radial bins. The latter has more potential and leaves room for generating bigger databases, so that I used it in all other and new experiments. All timings should be taken with a grain of salt, because some experiments have been made while I also used my computer and some experiments were measured (at night) without further interference. Nonetheless they should provide a good orientation.

Table 7 shows the results for the second and older generation of CAPTCHAs and table 8 for the (probably) fourth generation. All parameters for the word recognition have been adjusted and tested on the set from May 2010 and have been left unchanged for the other sets. It can be seen that these parameters generalize well across the different generations. For the fourth generation, the pruning was set to use the last character of a word. This was a quick measure to counter the use of upper case characters in the verification words of the new CAPTCHAs, due to the limited time left to analyze these new CAPTCHAs. No words with upper case letters have been recognized (the Levenshtein distance was calculated after converting these words to lower case), but recognitions are nonetheless remarkably good and are significantly better than for the third generation CAPTCHAs. A more sophisticated word recognizer for the fourth

<sup>&</sup>lt;sup>5</sup> http://www.keithv.com/software/wlist/

S	$l/\theta$ bins	D  <b>in GB</b>	$R_w$ total	$R_w$ to <b>D</b>	<i>R<sub>c</sub></i> (1)	n <sub>c</sub>	filter	$T_{avg}$	Lavg	P <sub>avg</sub>
fast	6/6	3.2GB	8.5%	14.7%	68.2%	1	no	1.6 s	4.50	2495
high	6/6	3.2GB	9.9%	17.1%	70.3%	1	no	8.1 s	4.31	2529.4
fast	6/6	3.2GB	10.5%	18.2%	85.3%	3	no	2.3 s	4.36	5877.7
high	6/6	3.2GB	11.7%	20.3%	84.6%	3	no	18.3 s	4.32	5774.8
high	6/6	3.2GB	12.7%	22%	90.2%	5	no	24.5 s	4.15	8269.7

Table 7: Final results for 496 second generation reCAPTCHAs obtained in November 2009

S	$l/\theta$ bins	D  <b>in GB</b>	$R_w$ total	$R_w$ to <b>D</b>	<i>R<sub>c</sub></i> (1)	n <sub>c</sub>	filter	$T_{avg}$	Lavg	$P_{avg}$
high	6/6	3.2GB	11.6%	23.5%	84.6%	3	no	15.4 s	4.30	5079.8
fast	6/6	3.2GB	9.6%	19.5%	83.9%	3	no	2 s	4.38	4993
high	6/6	3.2GB	10%	20.1%	88.6%	5	no	22.7 s	4.35	7410.3

Table 8: Final results for 301 fourth generation CAPTCHAs obtained on July 26, 2010.

generation of CAPTCHAs would however include some words with capital letters, for examples for some popular names and places. The fixed frame size was naively set to 40 pixel for the third generation and to 35 pixel for the second and fourth CAPTCHA generations, after measuring character widths on very few examples <sup>6</sup>.

It can be seen in all three different CAPTCHA sets that the number of best matching characters used for pruning,  $n_c$ , is a tradeoff between time, character accuracy and word accuracy. A higher number for  $n_c$  results in a high chance that the solution is not pruned away from the search space, but the word recognizer needs considerably more time to search in the bigger search space and the chance to find the correct word is smaller. If  $n_c$  is set to a smaller value, the resulting average size of the search space is logically much smaller and the performance and accuracy of the word recognizer is better as long as the solution is still in the search space, because fewer comparisons must be made. However, the probability that the solution is still in the search space is smaller then, so the that overall recognition rate can be smaller for to small values of  $n_c$ . In my experiments, a good value for  $n_c$  ranged between 3 and 5.

filing		but	ons	790n	ten	rebut		
	g	37639.4	S	40091.4	q	44235.9	t	39486.8
	s	47597.4	g	44122.9	a	47007.4	g	44258.2
	z	49626.2	a	44903.3	n	49750.9	f	45460.4
	p <b>iling</b>	3768.07	buttons	2929.18	women	2918.72	robot	3813.6
	differs	3814.65	bishops	3068.95	woman	3070.55	d <b>ebut</b>	3832.94
	sifting	3815.3	hostages	3124.58	worsen	3134.11	reliant	3889.19
	gliding	3822.83	britons	3155.62	carmen	3257.44	defeat	3904.02
	citing	3824.02	balloons	3156.59	seaman	3317.44	reflect	3927.76

## **Table 9:** Some individual recognitions for the second generation reCAPTCHAs. Single character pruning was used with<br/>the best three matches for the last character of the verification word. The three characters are shown and the<br/>five best matching words after pruning and better-half-search.

Table 9 lists some individual results for matches for the second generation reCAPTCHAs. The three best matching last characters and the final distances are listed. The search space was pruned with these characters and final word matches and their  $D_S$  distances are listed below the single character matches. Words with an Levenshtein distance of 0 or 1 have been highlighted. It is clearly visible that the character matching works very reliably if the last character is not too distorted and rotated. However, if it is too distorted, the best matching character might not be the best choice for pruning. For the verification word "woman" in this example it is "q", the good results from the word recognition would not be possible in this case if only "q" would have been used for pruning. However "n" is still among the best three matching character results and if they are all used for pruning, the word recognition can still find the best matching result and should recognize that "q" is very unlikely the correct last character of the word.

<sup>&</sup>lt;sup>6</sup> Indicated numbers refer to sizes used in a to 200% enlarged edge image

SI	Ekier	kier anto		ols set		ten	temps		atton	
r	34751.6	S	40060.5	d	43745.4	t	38663	t	34099.1	
f	38363.6	g	43565.2	g	47778.3	i	40085.8	i	36604.6	
р	47392.7	b	48938.2	р	49778.3	f	43765.8	1	38187.7	
cashier	4022.42	capitols	3874.07	sacrificed	4083.46	tempe	3571.55	artifa	act 4264.38	
prettier	4130.32	addicts	4038.95	serviced	4202.91	tampa	3725.03	amid	st 4356.63	
scatter	4185.19	casings	4106.19	specified	4448.69	tempo	3803.73	addio	ct 4379.27	
rectifier	4291.35	capitals	4132.76	stabilized	4459.23	tenure	3849.79	pacif	ist 4391.88	
shakier	4331.3	realizes	4185.48	accredited	l 4466.05	female	3927.01	apric	ot 4493.64	

Table 10: Some individual recognitions for the third generation reCAPTCHAs. Single character pruning was used with the<br/>best three matches for the last character or first character, depending on which on is farer away from the ellipse.<br/>If the correct character is not in the list of best matches, then the word recognition fails with high probability,<br/>this can be seen with the word "anion" here.

Table 10 shows some individual results for the third generation CAPTCHAs. Character widths are not equal in all cases, so that the naive assumption to segment one character with a fixed size fails on characters bigger than the average one. This is clearly visible with word "anion", where the final character "n" is bigger in width than the last characters in the other examples. Only half of this "n" was used for the character recognition, which in this case returned that "t", "i" or "l", are likely characters, although wrong for the last character of the word, these are still good matches for the right half of the character "n".

furnished		SOLLOW		janor		011001057		Mathews	
d	39261.4	b	48503.8	r	37731.3	у	39909.6	s	48309.6
g	43706.1	w	52960.4	n	47616.3	v	43041	g	49671.3
S	46517.6	v	52992.5	С	48287.6	g	47961.5	m	52100.7
furnishe	<b>d</b> 2902.55	sorrow	3423.41	tenor	3906.12	oncology	2803.96	bystanders	3280.19
tarnished	2914.82	narrow	3754.61	saner	3934.02	sociology	2938.2	apotheosis	3357.07
occasions	3167.82	sparrow	3788.07	japan	3954.13	radiology	2948.56	bulldozers	3363.4
ancestors	3187.41	glasgow	3839.14	s <b>ailor</b>	3978.74	archeology	2948.66	falsehoods	3364.32
exclusions	s 3190.25	marrow	3866.05	jaguar	4000.79	cardiology	2956.31	flashbacks	3376.49

Table 11: Some individual recognitions for the (probably) fourth generation reCAPTCHAs. Single character pruning was<br/>used with the best three matches for the last character of the word. The shape dictionary was generated entirely<br/>in lower case characters – it was initially just meant for the second and third reCAPTCHA generations – so that<br/>verification words with upper case characters are not recognized.

Table 11 shows some individual results for the fourth generation CAPTCHAs from reCAPTCHA. At the timing of writing, these are the CAPTCHAs deployed by reCAPTCHA. As one can see, the results are remarkably well even though the words are more distorted as in the other generations. This suggest that the deployed shape context word recognition is quite robust to such kinds of deformation, as no attempt was made to reverse them. Because no attempt besides choosing the last character for pruning was made to counter the multiplied use of names and upper case letters in this new generation, recognitions of such words and names failed to give good results, as it can be seen exemplary with the name "Mathews" in table 11.

#### 7.1 Practical considerations

For an adversary, the "fast" profile might be more appealing than the "high" one. Comparing numbers for both profiles with the fourth generation and a character pruning of three characters, the "fast" one is more than 7 times faster, sacrificing only 2% of the accuracy. In a practical setting, he would need to solve two words per CAPTCHA, because he does not know the position of the verification word. At a rate of about 10%, he could pass one CAPTCHA in about  $2 \cdot 10 \cdot 2 = 40$  seconds with the "fast" profile and one quad-core computer (with 3GHz). The same number for the "high" profile would

be about  $2 \cdot \frac{1}{0.115} \cdot 15.4 \approx 268$  seconds. There might be additional practical obstacles, like an IP banning from the reCAPTCHA server after to many failed tries from the same IP But, let our adversary be a powerful one, that can send requests from many different IPs, to counter such an additional security measure. Then, at a rate of one CAPTCHA per 40 seconds, he could pass about 2160 CAPTCHAs a day with the "fast" profile and about 322 with the "high" profile with one computer. The settings could be even more tuned for speed and throughput, but for a more power adversary this might not be a problem, he could simply use 100 instead of one computer. Then he could pass about 216000 CAPTCHAs from reCAPTCHA in one single day (with the "fast" profile).

#### 8 Conclusion

In this Bachelor thesis, I showed that it is indeed possible to write a software solver that can pass more than 5% of CAPTCHA challenges from reCAPTCHA. The proposed holistic word recognizer on shape context basis could be quickly adapted to changes in the deployed CAPTCHA of reCAPTCHA, works without learning and generalizes well over different recent reCAPTCHA generations. However, results were initially poor and below 5% for the second last generation of CAPTCHA deployed in the first half of 2010, due to additional distortions. A suitable preprocessing step is specially proposed for this sort of CAPTCHA. It uses machine learning and the additional distortions could be filtered, so that word recognition rates could be increased with this specialization to about 5%.

Rates are much higher for an older generation of reCAPTCHA, deployed until the end of 2009. With up to over 12%, measured on verification words only, the numbers compare favorably to an experiment made earlier by Wilkins with an OCR software in 2009 [58]<sup>7</sup>. Ironically, recognition rates are significantly higher for the newest deployed CAPTCHAs from reCAPTCHA compared to the previous one, with recognition rates up to above 11%. This number is in the range of the recognition rates for the 2009 era, although deformations in the words seem to be stronger and words and names with upper case letters are used. This gives further experimental proof that the used shape context recognition scheme is still robust in the presence of such stronger deformations.

It could be shown that a holistic word recognizer is an elegant solution to recognizing words that are difficult to segment into isolated characters. The results presented in this thesis mean for CAPTCHA designers, that even a segmentation resistant word CAPTCHA is not a guarantee that the CAPTCHA cannot be computationally solved by other means than segmentation. The security question of a text CAPTCHA can not be reduced to its segmentation resistance.

For the presented holistic word recognizer, I slightly changed the shape context matching framework proposed earlier by Mori et al. [43], used additional pruning methods with single characters and could show that a holistic recognizer based on object recognition is computational feasibility with a dictionary of about 22'000 words. However, the proposed recognizer is only a proof of concept and by using larger dictionaries or specializations for the latest generation of CAPTCHAs from reCAPTCHA, much higher recognition rates should be attainable. The matching framework by Mori et al. was also not fully explored, especially further and more sophisticated methods for the detailed matching have not been used. The relatively high memory needs of about 3 GB (and more for bigger word lists) could also be drastically improved by compressing the data and storing the histograms differently. The proposed better-half-search algorithm for searching the best match in a dictionary can be individually tuned to give higher accuracy or faster results. Recognition times in the range of 1 - 2 seconds per word are attainable with a slight loss in accuracy and could be additionally improved, so that this work could prove its usefulness in other situations of difficult to segment characters, for example in older documents with difficult typefaces or handwriting.

reCAPTCHA is considered to be one of the most difficult text CAPTCHAs and justly so. Writing a software solver for it proved to be quite a challenging task. But the results presented here show that it is not impossible to build a strong adversary for reCAPTCHA. The rates and figures obtained for all three different reCAPTCHA generations should also suffice for making practical exploitation feasible. With the definition that a CAPTCHA is effectively broken, if it can be computationally solved at a considerable small rate of 5% or more, all three generations tested in this bachelor thesis, from the 2009 era up to this date are effectively broken. The reCAPTCHAs with additional distortions proved to be the most challenging ones and provided a little more security over the currently deployed CAPTCHAs for a holistic recognizer, but a better preprocessing procedure could decrease this gap and could also be adapted to different additional distortions, if reCAPTCHA choses to reimplement this idea in a different fashion.

However, the system reCAPTCHA per se is not broken, because the deployed CAPTCHA could be quickly adapted to counter a new form of attack, like this holistic one. For example, all words that my holistic recognizer on shape context basis can solve, could be sorted out of the verification words pool. But the general feasibility of using a holistic recognizer (not a particular one), requires rethinking beyond traditional OCR software with segmentation algorithms for the designers of reCAPTCHA. Eventually, it could prove quite difficult to design a new word CAPTCHA that counters a holistic attack, while still keeping overall usability for humans (and human recognition rates) on the same level by using English words.

Obviously, the gap between human recognition rates and the rates presented in this bachelor thesis is still a wide one. Further research could decrease this gap, up to the point there machines could be considered to be better readers than humans. It will be impossible to increase the difficulty of text CAPTCHAs forever, as a CAPTCHA should remain easily solvable by humans. Thus, new unsolved AI problems should be explored for CAPTCHAs, so that secure CAPTCHAs can still be build, when one day the AI-problem of reading (distorted) words is completely solved.

<sup>&</sup>lt;sup>7</sup> However, Wilkins measured final recognition rates on both verification and read words. Up to this date, read words are irrelevant for passing a reCAPTCHA, but the two groups of words are of different difficulty; so that it is not fair compare the numbers 1:1

#### 9 Acknowledgements

Special thanks go to Niklas Büscher, with whom I developed the idea of holistic word recognition for the second generation reCAPTCHAs in a practical course on CAPTCHAs at the Technische Universität Darmstadt (Darmstadt University of Technology) in the end of 2009. The extended and ambitious software solver in this thesis would not have been possible without these first practical results and studies of feasibility. I want to thank family and friends, especially David Pietrzik for assistance in labeling second generation reCAPTCHAs at the end of 2009.

Also, I acknowledge gratefully Dr. Marc Fischlin and Paul Baecher for their supervision of my thesis and many good remarks and comments. I particularly enjoyed the freedom they gave me in my research. Furthermore I want to thank Dr. Marc Fischlin for making an interesting practical course on CAPTCHAs available at our university, which I can highly recommend to my fellow students. I specially thank him for his offer to extend the practical work from 2009 in a bachelor thesis.

#### References

- [1] AS El Ahmad, J Yan, and L Marshall. The robustness of a new CAPTCHA. *Proceedings of the Third European Workshop on System Security*, 2010. 2.2, 3.1
- [2] L Von Ahn, B Maurer, C McMillen, D Abraham, and M Blum. recaptcha: Human-based character recognition via web security measures. *Science*, 2008. 1, 2, 2.2, 3, 4, 4.3.3, 4.3.4
- [3] U.H. Ai, L. Von Ahn, M. Blum, and J. Langford. CAPTCHA: Using Hard AI Problems For Security. In *Proceedings of Eurocrypt*, pages 294–311. Citeseer, 2003. 1, 3
- [4] M Alan. Turing. Computing machinery and intelligence. Mind, 1950. 1
- [5] HS Baird and TP Riopka. ScatterType: A reading CAPTCHA resistant to segmentation attack. Proc. SPIE, 2005. 3
- [6] DM Baxter. Surface Dyslexia: Neuropsychological and Cognitive Studies of Phonological Reading. British Medical Journal, 1987. 3.2
- [7] BBC. Spam weapon helps preserve books. 1
- [8] S Belongie, G Mori, and J Malik. Matching with shape contexts. *Statistics and Analysis of Shapes*, 2006. 1, 3.1, 6.2, 6.3, 6.3.1, 6.3.2, 6.5.2
- [9] Blackwidows. Breaking CAPTCHAs (as accessed on Jul 23 2010). 3
- [10] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly Media, 2008. 5.3, 5.5
- [11] T.M. Breuel. The OCRopus open source OCR system. In Proc. IS&T/SPIE 20th Annual Symposium 2008. Citeseer, 2008. 3.1
- [12] J Canny. A computational approach to edge detection. *Readings in computer vision: issues, problems, principles, and paradigms*, 1987. 5(b), 4.3.1, 5.1, ??, 6.4
- [13] JMK Cattell. The time taken up by cerebral operations. *Mind*, 1886. 3.2
- [14] K. Chellapilla, K. Larson, P. Simard, and M. Czerwinski. Computers beat humans at single character recognition in reading based human interaction proofs (HIPs). In *Proceedings of the Second Conference on Email and Anti-Spam*, pages 21–22. Citeseer, 2005. 1, 3.1, 1
- [15] K. Chellapilla, K. Larson, P. Simard, and M. Czerwinski. Designing human friendly human interaction proofs (HIPs). In *Proceedings of the SIGCHI conference on Human factors in computing systems*, page 720. ACM, 2005. 1
- [16] K. Chellapilla, K. Larson, P.Y. Simard, and M. Czerwinski. Building segmentation based human-friendly human interaction proofs (HIPs). *Human Interactive Proofs*, pages 1–26, 2005. 2.2
- [17] K. Chellapilla and P. Simard. Using machine learning to break visual human interaction proofs (HIPs). Advances in Neural Information Processing Systems, 17, 2004. 1, 3.1
- [18] M Cheriet, N Kharma, CL Liu, and C Suen. Character recognition systems: a guide for students and practioners. 2007. 3

- [19] M Cheriet, N Kharma, CL Liu, and CY Suen. Character Recognition Systems: A guide for students and practitioners. 2007. *Wiley, New York*. 4.3
- [20] M Chew and JD Tygar. Image recognition captchas. Information Security, 2004. 2.1
- [21] AL Coates, HS Baird, and RJ Fateman. Pessimal print: a reverse Turing test. *Proceedings of the Sixth International Conference on Document Analysis and Recognition*, 2001. 1, 3
- [22] Y Freund and R Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. *Computational learning theory*, 1995. 5.3
- [23] Y Freund and RE Schapire. Experiments with a new boosting algorithm. *Machine learning-international workshop then conference*, 1996. **5**.3
- [24] R Frost. Toward a strong phonological theory of visual word recognition: True issues and false trails. *Psychological Bulletin*, 1998. **3**.2
- [25] H Fujisawa. Forty years of research in character and document recognition-an industrial perspective. *Pattern Recognition*, 2008. 1, 3
- [26] GoogleBlog. Teaching computers to read: Google acquires reCAPTCHA, 2009. 4
- [27] J Grainger and AM Jacobs. Orthographic processing in visual word recognition: A multiple read-out model. *Psy-chological review*, 1996. 3.2
- [28] Paul W. Handel. United States Patent 1915993 Statistical machine, 1933. 1
- [29] SY Huang, YK Lee, G Bell, and Z Ou. A projection-based segmentation algorithm for breaking MSN and YAHOO CAPTCHAs. *Proceedings of the World Congress on Engineering*, 2008. 3.1
- [30] Johnny Ling. Number of Words in the English Language, 2001. 3.3
- [31] R Jonker and A Volgenant. A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing*, 1987. 6.3.1
- [32] AL Koerich, R Sabourin, and CY Suen. Large vocabulary off-line handwriting recognition: A survey. *Pattern Analysis* & *Applications*, 2003. 3.3
- [33] M Körtgen, GJ Park, M Novotni, and R Klein. 3D shape matching with 3D shape contexts. *The 7th Central European Seminar on Computer Graphics*, 2003. 3.1
- [34] H Krim and AJ Yezzi. Statistics and analysis of shapes. 2006. 3.1
- [35] H.W. Kuhn. The Hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955. 6.3.1
- [36] V Lavrenko, TM Rath, and R Manmatha. Holistic word recognition for handwritten historical documents. *First International Workshop on Document Image Analysis for Libraries, 2004. Proceedings, 2004.* **3**.2
- [37] MD. Lillibridge, M. Abadi, K. Bharat, and A. Border. Method for selectively restricting access to computer systems. US Patent 6,195,698, 2001. 3
- [38] Nicholas J. Hopper Luis Von Ahn, Manuel Blum and John Langford. the official captcha site, 2000. 1, 3
- [39] D.J.C. MacKay. Information theory, inference, and learning algorithms. Cambridge Univ Pr, 2003. 6.4.1
- [40] S Madhvanath and V Govindaraju. The role of holistic paradigms in handwritten word recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2001. 3.3
- [41] UV Marti and H Bunke. Using a statistical language model to improve the performance of an HMM-based cursive handwriting recognition system. *International Journal of Pattern Recognition and Artificial Intelligence*, 2001. 3.3
- [42] Merriam-Webster. Merriam-Webster's Manual for Writers and Editors. 1998. 4.3

- [43] G Mori, S Belongie, and J Malik. Shape contexts enable efficient retrieval of similar shapes. 2001. 3.1, 6.2, 3, 6.3, 8
- [44] G Mori, S Belongie, and J Malik. Efficient shape matching using shape contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2005. 3.1, 6.3.3
- [45] G Mori and J Malik. Estimating human body configurations using shape context matching. Computer Vision—ECCV 2002, 2002. 3.1
- [46] G Mori and J Malik. Recognizing objects in adversarial clutter: Breaking a visual captcha. 2003. 3.1, 3.3
- [47] M. Motoyama, K. Levchenko, C. Kanich, D. McCoy, G.M. Voelker, and S. Savage. Re: CAPTCHAs–Understanding CAPTCHA-Solving Services in an Economic Context. *cseweb.ucsd.edu*. **3**.1
- [48] G Moy, N Jones, C Harkless, and R Potter. Distortion estimation techniques in solving visual CAPTCHAs. 2004. 3.1, 3.3
- [49] G Nagy, TA Nartker, and SV Rice. Optical character recognition: An illustrated guide to the frontier. *Proceedings of SPIE*, 1999. **3**
- [50] M Perea and E Rosa. Does whole-word shape play a role in visual word recognition? *Perception & Psychophysics*, 2002. 3.2
- [51] DC Plaut and T Shallice. Deep dyslexia: A case study of connectionist neuropsychology. *Cognitive neuropsychology*, 1993. 3.2
- [52] P.Y. Simard, R. Szeliski, J. Benaloh, J. Couvreur, and I. Calinov. Using character recognition and segmentation to tell computer from humans. *Document Analysis and Recognition*, 1:418, 2003. 3
- [53] J. Tam, J. Simsa, S. Hyde, and L. Von Ahn. Breaking audio captchas. Advances in Neural Information Processing Systems, 1(4), 2008. 2.2
- [54] O Taylor. Pango: internationalized text handling. linuxsymposium.org. 7
- [55] A Vinciarelli, S Bengio, and H Bunke. Offline recognition of large vocabulary cursive handwritten text. *Document Analysis and Recognition*, 2003. 3.3
- [56] P Viola and M Jones. Fast and robust classification using asymmetric adaboost and a detector cascade. Advances in Neural Information Processing Systems, 2002. 5.3
- [57] DD Wheeler. Processes in word recognition. Cognitive Psychology, 1970. 3.2
- [58] J Wilkins. Strong captcha guidelines v1. 2. 2009. 3.1, 8
- [59] J Yan and AS El Ahmad. A Low-cost Attack on a Microsoft CAPTCHA. Proceedings of the 15th ACM conference on Computer and communications security, 2008. 1, 3.1
- [60] K Zimmermann and T Svoboda. Approximation of euclidean distance of point from ellipse. *European Medical and Biological Engineering Conference*, 2005. **5.4.1**