CZECH TECHNICAL UNIVERSITY IN PRAGUE
FACULTY OF INFORMATION TECHNOLOGY

ASSIGNMENT OF BACHELOR'S THESIS

Title:	Pattern recognition of CAPTCHA
Student:	Matouš Pištora
Supervisor:	Ing. Martin Kopp
Study Programme:	Informatics
Study Branch:	Computer Science
Department:	Department of Theoretical Computer Science
Validity:	Until the end of winter semester 2017/18

Instructions

The main goal of this work is automatic recognition of characters generated by various CAPTCHA solutions, using computational intelligence. The purpose is to show weak spots of such systems. Student will implement the Canny edge detector and an algorithm for image segmentation. Single characters will be recognized using a trained k-nearest neighbor classifier. This will produce a solution with a CAPTCHA image on the input and the most probable set of characters on the output.

References

CAPTCHA: Using Hard AI Problems for Security
http://repository.cmu.edu/cgi/viewcontent.cgi?article=1142&context=compsci

Text-based CAPTCHA Strengths and Weaknesses
http://www.cin.ufpe.br/ rsc3/temp/text-based-captcha-strengths-and-weaknesses.pdf

The End is Nigh: Generic Solving of Text-based CAPTCHAs
https://www.usenix.org/system/files/conference/woot14/woot14-bursztein.pdf

Use of the Hough Transformation to detect lines and curves in pictures
http://www.dtic.mil/dtic/tr/fulltext/u2/a457992.pdf

Canny Edge Detection Enhancement by Scale Multiplication
ftp://openstorage.gunadarma.ac.id/research/IEEE/PAMI/Sep_05/i1485.pdf

L.S.

doc. Ing. Jan Janoušek, Ph.D.
Head of Department

prof. Ing. Pavel Tvrdík, CSc.
Dean

Prague February 24, 2016

Czech Technical University in Prague
Faculty of Information Technology
Department of Theoretical Computer Science

Bachelor's thesis

Pattern Recognition of CAPTCHA

Matouš Pištora

Supervisor: Ing. Martin Kopp

17th May 2016

Acknowledgements

I would like to express my gratitude to my supervisor, Ing. Martin Kopp, for his insight and guidance throughout writing this thesis. I would also like to extend my thanks to my parents for their constant encouragement and help.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on 17th May 2016 ....

Citation of this thesis

Pištora, Matouš. *Pattern Recognition of CAPTCHA*. Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2016.

Abstrakt

Tato práce se zabývá automatickým rozpoznáváním znaků generovaných CAPTCHA řešeními. Jejím cílem je poukázat na nedostatky a slabá místa v jejich návrzích. Studovány a vyhodnocovány jsou různé prvky návrhů. Je vytvořen a realizován algoritmus pro segmentaci obrazu. Jednotlivé znaky jsou klasifikovány pomocí algoritmu strojového uení. Výsledný algoritmus je nasazen na třech řešeních v několika experimentech.

Klíčová slova bezpečnost webových stránek, CAPTCHA, optické rozpoznávání znaků, počítačové vidění, strojové učení

Abstract

This thesis studies automatic recognition of characters generated by CAPTCHA solutions. Its aim is to point out the flaws and weak spots in their design. Various design features are studied and evaluated. An algorithm for image segmentation is created and implemented. The individual characters are classified using a machine learning algorithm. The resulting algorithm is deployed on three solutions in several experiments.

Keywords CAPTCHA, machine learning, optical character recognition, computer vision, website security

Contents

List of Figures

List of Tables

Introduction

In the past few decades, the rise of Internet has revolutionized our lives. We use it for studies, work, socializing, shopping and many other activities on a daily basis. With the increasing popularity of the web though, many public services have been targeted by malicious attackers. Among other activities, they attempt to exploit mail servers for sending massive amounts of spam messages, create numerous fake profiles on social networks or make fraudulent offers on online marketplaces. In order to block the access of automated scripts and bots, websites started to use various security measures so as to ensure their safety. One type of such tools is the CAPTCHA[1]. The acronym stands for "Completely Automated Public Turing Test to Tell Computers and Humans Apart". As the attackers tried to overcome the security measures, the website owners were forced to deploy increasingly more difficult challenges, and the battle has eventually developed into an arms race between the two sides. Its benefit, on the other side, is that it pushes the boundaries of artificial intelligence and optical character recognition.

The main goal of this thesis is the automatic recognition of characters generated by captcha solutions. Using computational intelligence and a custom segmentation method, an algorithm will be implemented taking an image with a captcha challenge on the input, and resulting in a most probable set of characters on the output.

An additional goal of this thesis is to study captcha design features. A thorough research will be undertaken, and based on acquired knowledge and experiments, their strengths and weaknesses will be evaluated and their design flaws exposed.

This thesis consists of six main chapters. The first chapter gives an insight on current captcha schemes and studies and evaluates their various design features. The second chapter presents an overview of the algorithm which is later used to break the captcha. The third chapter deals with the pre-

[1] I will use lower case "captcha" in future reference for typographical reasons

processing part of the algorithm pipeline. The fourth chapter proposes a segmentation algorithm which is used to isolate individual characters. The fifth chapters deals with the recognition of the characters achieved by methods of computational intelligence, specifically machine learning algorithms. The sixth chapter summarizes the results of my experiments with the implemented algorithm. The last chapter concludes the thesis.

Chapter 1

Analysis

1.1 Captcha

The acronym captcha was coined in 2003 by von Ahn et al[1]. In this paper they present the basic idea of captcha and introduce a definition of what it exactly is. For the exact definition I refer the reader to the original paper. The fundamental idea is to use a useful and yet unsolved hard AI problem which is easy for humans to solve. It implies a win-win situation in which either the problem is unsolved and we have an effective way to discern humans and robots or a solution appears which develops the field of AI problems. It is the main motivation for majority of researchers working in this branch of computer science.

A captcha is a program that generates and then grades a test which the majority of humans are able to solve, but current computer programs cannot. Its main application is on websites where it is used in order to distinguish whether the user is a human or a robot. The need for this type of challenge arose with the increasing amount of Internet bots and automated scripts attempting to exploit public web services. Nowadays it is an established security mechanism to prevent mailing spam messages, mass posting on internet forums, mass voting in online polls and downloading files in massive amounts.

In artificial intelligence, standard Turing test [2] is defined as a test in which a human judge is supposed to consistently distinguish whether he/she is communicating via text with a human counterpart or a computer pretending to be a human. However on the Internet for such a test to be used automatically and effectively the judge must be also a computer. This is where captcha comes into play. It is also sometimes described as a “reverse Turing test”, because the judge is not a human but a computer. This term can be however misleading, because it can also imply a test in which both human and computer participants are attempting to prove that they are the computer, not the human.

Another significant work was done by Microsoft researcher Chellapilla[3] who calls these tests Human interaction proofs (HIPs for short). His work focuses on distinguishing effective distortion features and specifying best practices for designing captchas which are resistant to computers while remaining relatively easy for humans to solve. He also states that, depending on the cost of the attack, automated scripts should not be more successful than 1 in 10 000 attempts, while human success rate should approach 90%. It is generally considered a too ambitious goal, as random guesses can be successful[4], and subsequently a captcha is considered compromised when the attacker success rate surpasses 1%.

1.2 Types of captcha

Over the years there have been developed many types of captcha. In the following section is the brief survey of current captcha schemes.

1.2.1 Text-based schemes

The first captcha ever used was in 1997 by the software company Alta-Vista which sought a way to prevent automated submissions to their search-engine. It was a simple text-based test which was sufficient for the time, but it was eventually proven ineffective. At the time the computer recognition rates of single characters were at par with those of humans[4] and thus the development of captchas shifted to prevention of segmentation like noise addition, cluttering and other various anti-segmentation techniques. As the popularity of the Internet rose, so did the amount of malicious activities and accordingly the popularity of captchas. With the effort to prevent breaking of captchas with increasing the amount of distortion and cluttering, the challenges faced the risk of becoming almost illegible. The design of captchas developed into a challenging task in order for them to be human friendly and secure at the same time.

1.2.2 Sound-based schemes

From the beginning the adoption of captcha schemes was not an ideal state. The users were annoyed with the bothersome challenges that were at times almost illegible and had to try numerous attempts in order to solve them. The people affected the most were the ones with visual impairments or various reading disorders such as dyslexia. For them surfing the Internet was already a very challenging task and this obstacle made the web even more inaccessible. But soon an alternative emerged in the form of audio captchas. Instead of looking at the image and transcribing the displayed characters, the user was given the option, usually alongside with a traditional text-based captcha, to play a sound puzzle and write the characters that he heard. In order to remain effective and secure the captcha has to be resistant to automated sound analysis. For this purpose various background noise and sound distortion were added. Still a human visitor should have no problem in hearing and recognizing the code. Generally, this scheme is now a standard option on major websites who implement captcha.

1.2.3 Image-based schemes

With the rise of the captcha schemes criticism soon began to appear. The obstacle of solving a puzzle every time someone wants to enter a site is at least annoying and discouraging for the common user. On the other hand, the companies who owned the websites were unsatisfied as well. Imagine a customer visiting an online shop who is presented with a captcha challenge, fails to solve it and switches to a competitor. It is in a company's best interest to keep the customer satisfied all the time and make their user experience the most pleasant. In order to preserve security against spam-bots, new captcha designs were developed. Ranging from various forms of video captchas to miniature puzzle games and math questions, the most prominent design was image-based captcha. The user is presented with a series of images showing various objects and the task lies usually in detecting which of them have a common topic and selecting them. For example a user is shown a series of images with various animals and is supposed to select the ones with a cat. This type of captcha has gained huge popularity on smart phones, where simply tapping the screen is the preferable option over typing the code.

1.3 Anti-recognition features

Character recognition has long been one of the major problems in the field of artificial intelligence. It has been vastly explored and modern classifiers perform very well on character recognition (up to 99.5% on a handwritten digit dataset[4]). Accordingly, we cannot rely on anti-recognition techniques to prevent breaking the captcha, but we rather use them so as to increase security.

1.3.1 Set of characters

The volume of the character set (i.e. using only digits, alphabetical letters or a combination of both) can have a huge impact on both the computer classifier and the person attempting to solve the captcha. Increasing it reduces the attacker's success rate by decreasing the classifier performance and it becomes more resistant to brute-force attacks by randomly guessing the answer. However when the attacker expands the set, on which the classifier is learning, it negates the effect, because the recognizer can learn all the characters at the cost of taking more time in the learning phase. On the other hand it also reduces the human accuracy by introducing easily confusable characters for example i and j or l an I. All in all the reduced readability is too significant compared to the small security gain and so the recommendation is to use a small non-confusable set of characters.

1.3.2 Multiple fonts

Another similar technique is to use multiple fonts. It is no problem for humans to read letters in different fonts as we do it on everyday basis. This technique by itself is not much different from increasing the charset, but with an introduction of an anti-segmentation technique changing the fonts makes the size of the letters unpredictable and in turn makes this technique relatively effective.

1.3.3 String length

Length of the string in each challenge is very important. Along with the character set size, it is the basis for resisting brute-force attacks. The more characters the captcha contains, the harder it is to guess or recognize the text. Assuming the rate of recognition of one character is $r, r < 1$, then the recognition rate of the whole challenge is r^n, where n is the length of the captcha. Note that the rate is exponential and it decreases as n grows. However the length of captchas has uncertain implications. A fixed length benefits the user in his recognition efforts, as it helps him discern clutter from easily confusable letters such as confusing a random vertical line with the letter 'l'. It can also help him recognize merged characters from single characters to match the length of the captcha, for example 'rn' and 'm', 'cl' and 'd'. On the other hand, randomizing the length of the captcha has huge drawback in the attacker's efforts as it is a vital information for segmentation algorithms. It is recommended to use medium to high and random length of captchas.

1.3.4 Dictionary words

The choice between random strings and dictionary words is not obvious. Although using dictionary words has been exploited to attack captchas[5], it has many benefits for the user and the idea should not be wholly abandoned. For people who understand the language, it is much easier to solve heavily distorted words than individual characters, because they can place the particularly heavily distorted letter into context of the whole word and try to guess it. It is upon the designer to consider whether to implement this feature or not.

1.3.5 Distortion

Distortion is a technique in which ripples and warp are added to the image. It is one of the easiest and most effective ways of reducing the classifier accuracy and scheme learnability. It can have a large impact on the user's accuracy though, as it can make the captcha particularly hard to read. Human captcha readability is a requirement of captcha design and thus usage of this feature cannot be recommended.

1.3.6 Rotation, slanting, sloping, scaling

Rotating, slanting or other spatial deformation of individual characters or the whole captcha does not imply decreasing the classifier performance, since using the right tools can revert or overcome the process. However, the use of anti-segmentation techniques makes the position and size of each character unpredictable which is effective in reducing the attacker's chance of success. Humans are altogether able to overcome these features and do not have major difficulties with them which makes the above specified tools a good technique to implement.

1.4 Anti-segmentation features

As previously mentioned, anti-recognition features do not by themselves guarantee the captcha safety, but rather help slow down the attacker and reduce their accuracy. The core defence mechanism has thus shifted to anti-segmentation features trying to prevent the attacker from dividing the text into isolated letters and recognizing them later, as it is considerably easier than recognising the text as a whole. A short summary of the most frequently used follows.

1.4.1 Confusing background

The idea of this feature is to use complex and confusing background such as a random image to hide the letters and prevent the attacker from isolating and recognizing them. Nevertheless, this idea is fundamentally flawed, because the user has to be able to decipher the text. The designer is forced to make the letters somewhat contrasting from the background. This in turn enables the attacker the isolate the captcha. That is why this technique is nearly useless and not recommendable.

1.4.2 Noise

A very common technique is to use random noise and clutter to confuse the segmentation. It is very easy to implement and thus can be found in many captchas. However, it is also quite easy to overcome. Many techniques for removing noise have been explored, analysed and successfully implemented, for example standard Gaussian smoothing with thresholding or a standard filter from image processing called *opening*[2] can easily erase most of the small

[2]More about this technique in the *Pre-processing*3 section of this work

clutter. We cannot therefore recommend to rely on this technique as the main anti-segmentation tool, but it can serve as a good complement.

1.4.3 Lines

Confusing lines and arcs in the scheme design can be implemented in various ways. It is vital to use the same color as the text, because the alternative would provide an easy way for the attacker to tell the lines and the text apart and render the use of them useless. Another important factor is to use the same line width as the characters, since doing otherwise would make them susceptible to denoising techniques mentioned previously. A good way of implementing them is to use lines which intersect some of the letters. Doing so strongly decreases the chance of good segmentation. A good implementation of occluding lines is one of the most effective and human-friendly ways of preventing segmentation and thus it is highly recommended.

1.4.4 Collapsing

Collapsing or negative kerning is a technique where neighboring letters are moved closer to each other in order to remove the spaces between them and form a continuous area. A form of collapsing can be the aforementioned rotating of characters, which possibly makes the letters touch and overlap one another. This makes it really hard to predict where the segmentation of the captcha should be done in order to obtain isolated letters. Even so there are a few design flaws to be avoided. If the length of the captcha is predetermined and the letters have a constant width, we can make the cuts based on an educated guess. Another thing to be aware of is that excessive collapsing [4] rapidly decreases human ability to solve the captcha and promptly makes it almost impossible to decipher. Properly implemented collapsing alongside with intersecting lines are the most recommendable anti-segmentation features.

1.5 Tested solutions

1.5.1 ČÚZK

The site “cuzk.cz”[6] is the official website of the Czech “State Administration of Land Surveying and Cadastre”. It enables its users to look at the cadastre map of the Czech Republic, find the information about a particular land parcels. Some of its other services, such as finding a property with the owner’s name, is however charged. This could be circumvented by downloading all the data and searching through them directly. Exploiting the database is nonetheless against the site’s terms and conditions and using its data can be against the law. In order to prevent access to automated bots, the website utilizes a captcha to prevent the mass-downloading of the database.

The design of the captcha is fairly standard. It contains characters of about five different fonts including serif and sans serif ones with various stroke width and character height and width. The characters *0* and *i* are missing so as to reduce their substitution with similar characters. The captcha contains both upper and lower case letters and Arabic numerals. A small global warp is applied and the letters have a variable horizontal and vertical offset. The letters usually do not touch the border of the image. In some cases, when the captcha is long, the last letter exceeds the right border so that up to a half of the letter is missing. In rare instances, the height of some letters and their vertical offset are so big that they touch the top or bottom border. The background is plain white and all the other features and font color are grey. It includes random noise with a grain of about four pixels. The main anti-segmentation feature is a curved line originating in the top left corner and curving throughout the image. The path of the line is very random, sometimes intersecting all the letters and other times going out of the image and returning to touch only a few letters. Occasionally, it does not touch any characters at all, not helping secure the captcha whatsoever.

The length of the captcha is mostly the same, but it sometimes varies. Based on my experience it depends on the amount of accesses from the same network. On a private home computer, the length of the captcha is five characters, but on a busy network, such as in school, the length is six characters. Nevertheless, the dimensions of the image remains the same. In the five character version the characters are usually isolated and touch only occasionally. In the longer version though, the characters touch very frequently, often pass through one another and occasionally completely cover the next letter. This reduces the human readability very significantly
and sometimes renders it impossible to solve.

1.5.2 mojedatovaschranka

As a part of modernizing bureaucratic apparatus, the Czech government introduced a system of online data repositories for exchange of documents between various government offices, companies and citizens[7]. It offers an alternative for delivering letters and makes it more effective. To sign in, one has to fill in his username and password and enter a captcha.

Comparing to the previous scheme, this one is fairly simpler. It shows yellow numbers on a white background with numerous curved lines of various width. However, the color of the lines is grey and they do not interfere with the individual characters in any way. The length of the captcha is constant, because it contains always five numerals. A global warp is applied to the image with the consequence that the letters have an irregular height and width and a slight deformation. In some instances it causes the letters to graze one another. Very rarely the letters touch the bottom or the top border.

If the user decides that the captcha is too hard to solve, an option to load a different challenge is offered. The site tracks the number of captchas the user has requested. After five attempts it shows the user an error message and refuses to load another one. This number is stored in the site's local cookies. However, the website has no control what happens to them so they can be easily erased. This makes this feature useless and only bothersome for the user.

1.5.3 ulozto

The website ulozto.cz[8] is a czech server dedicated to uploading, downloading and sharing various data files. The option of a sound-based captcha challenge is offered. The captcha on this site is harder than the previous ones. It includes four characters with either lower or upper case and the same font. No numbers appear in this scheme. The design features include uniform background noise in form of little dots and numerous occluding curved lines intersecting characters and each other and thus making the segmentation very hard. A global and local warps are applied varying the stroke width significantly. However, the individual characters never touch each other. This fact could be taken advantage of for possible segmentation.

Chapter 2

Algorithm overview

2.1 Outline

In this chapters and the following ones, I will summarize the main ideas of the algorithm and present a pipeline with which I will attempt to break the aforementioned schemes. Currently, there are numerous techniques to overcome the captcha security. The most common and also reliable performing is the segment-then-recognize approach consisting of two parts. In the first one, we will try to overcome the anti-segmentation techniques the captcha has. The image is converted to a binary pixel matrix and then divided into individual letters. It is usually achieved with image processing tools and a custom built algorithm depending on the implemented techniques. A machine learning algorithm to classify the individual letters will be used in the second part. The advantages are its simplicity, speed, and accuracy. However, it is customized for the targeted scheme and subsequently does not perform on all captchas but only similar ones. This can be considered as the main disadvantage.

A next notable approach is to find all possible cuts in the captcha, then construct every possible segmentation and find the one which globally maximizes the recognition rate. This is very advantageous, because there is no need to recognize which cuts are between the individual characters and which are not, because all of them are processed in the recognition phase. Only the most probable answer is then chosen.

Another method is to treat the captcha as a whole and not to bother with the segmentation. That is certainly a great convenience but constructing an algorithm which is capable of recognizing characters in a distorted image is a very difficult task. In recent years there has been a development in this area using neural networks, which are yielding increasingly better results.

2.2 Pipeline

The approach of my choice is the first one since it is a simple yet proven method which has good options for customization. The general pipeline of the algorithm is as follows:

1. **Pre-processing:** The image is converted to a greyscale image and then to a binary matrix. Most of the distortion and noise is also removed.

2. **Segmentation:** The characters in the image are identified and isolated. This step can produce multiple possible segmentations.

3. **Post-segmentation:** The isolated characters are sanitized of the remaining noise and resized in order to standardize them for the classifier algorithm.

4. **Recognition:** A learned classifier is used to recognize all the isolated characters.

5. **Post-processing:** Of all possible answers, the one maximizing the recognition rate is chosen. It is most likely to be the correct answer.

Chapter 3

Pre-processing

3.1 Image conversion

In this step, the color image is converted to a greyscale one. A colorful image of a common quality uses 8 bits per pixel per channel, which sums to a total 24 bits for RGB images. Nonetheless, a greyscale image needs only one brightness channel so only a third of the memory is used. This in turn saves us computational time. The image is converted from RGB colorspace to a greyscale space with the following equation[9] issued in a *Rec. 601* standard

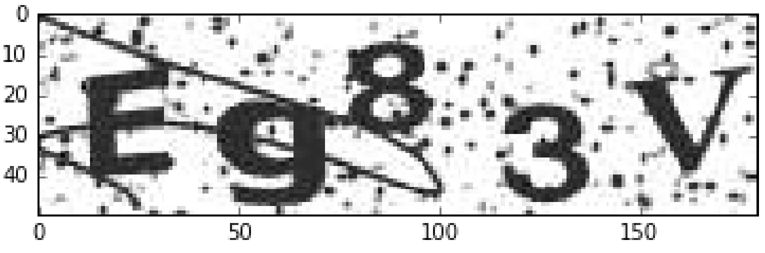


Figure 3.1: An example of a cuzk captcha with characters *Eg83V*

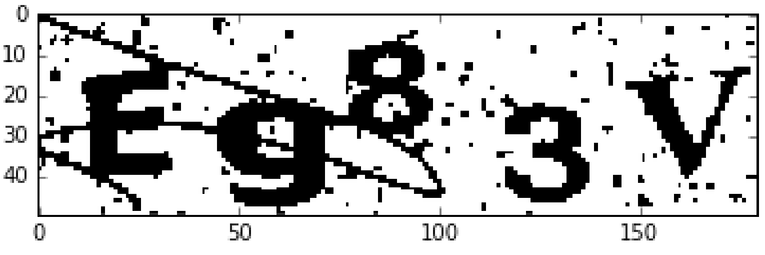


Figure 3.2: The thresholded captcha example

by the International Telecommunication Union[10]:

$$\text{RGB[A] to Gray:} \quad Y \leftarrow 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B \tag{3.1}$$

It is then further transformed with a thresholding method to a binary image. Thresholding is a technique where each pixel is assigned a new value: pixels with an intensity higher than the threshold are converted to white color and those with a lower intensity are converted to black. For a given threshold T the equations is:

$$f(x) = \begin{cases} 0 & \text{if } x < T \\ 1 & \text{otherwise} \end{cases} \tag{3.2}$$

This allows for further optimization in memory management as each pixel now theoretically uses only one bit. In my experiments, the threshold value can be set with two approaches. The first one is to empirically determine the threshold by analysing the samples and manually set the threshold value which yields the best results. The second approach is to use an automated algorithm for each captcha. The method of my choice is the one proposed by Otsu[11]. The threshold is computed by iterating through all possible thresholds and selecting the one which minimizes the within-class variance defined as a weighted sum of variances of the two classes. The class probabilities used as weights and the class variances are computed from the image brightness histogram:

$$\sigma_w^2(t) = \omega_0(t)\sigma_0^2(t) + \omega_1(t)\sigma_1^2(t) \tag{3.3}$$

$$\omega_0(t) = \sum_{i=0}^{t-1} p(i) \tag{3.4}$$

$$\omega_1(t) = \sum_{i=t}^{L-1} p(i) \tag{3.5}$$

3.2 Noise removal

The following section outlines the next step of the algorithm. It consists of removing the noise from the image.

3.2.1 Morphological operations

The next step is to remove the speckles and the occluding line from the image. One simple approach to remove speckles and occluding lines is to use morphological operations. They are a very powerful tool in the field of image

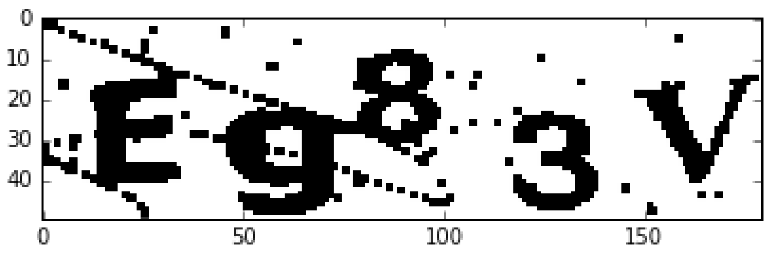


Figure 3.3: The effect of morphological *closing* with one iteration

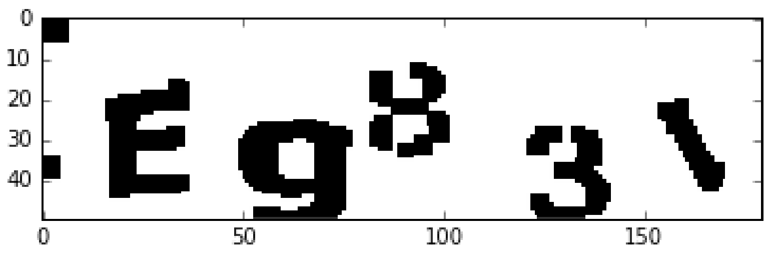


Figure 3.4: Deterioration of character details after three iterations of *closing*

processing and mathematical morphology. With the closing operation, we can fill small holes and gaps in the image, and with the opening operation loosely connected segments are disjointed and small points and lines are removed. In order to separate the individual letters, we choose the second operation. The four basic binary morphological operations of dilation $\oplus$, erosion $\ominus$, opening $\circ$ and closing $\bullet$ are defined as follows:

$$X \oplus H = \{(x, y) : H_{(x,y)} \cap X \neq \emptyset\} \tag{3.6}$$

$$X \ominus H = \{(x, y) : H_{(x,y)} \subseteq X\} \tag{3.7}$$

$$X \circ H = (X \ominus H) \oplus H \tag{3.8}$$

$$X \bullet H = (X \oplus H) \ominus H \tag{3.9}$$

where X is the original image, H the structuring element and $H_{(x,y)}$ the translation of H by the vector (x, y). The effect of closing operation can be described as erasing the object border and then regrowing it back. If in the first step an object is small enough to be considered a border as a whole, there is subsequently nothing to regrow and thus it is deleted. While this approach should remove all the speckles effortlessly, it can also destroy significant features of certain characters such as serifs or the i dots. Therefore the usage of this method is upon consideration.

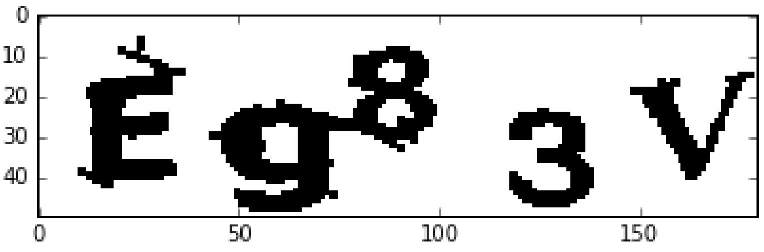


Figure 3.5: Removal of speckles with the flood fill algorithm

The next approach is to count the area of all connected components (i.e. how many pixels it contains) and delete the ones with area under certain threshold. The idea is to iterate on each pixel of the image and when a white pixel is found a flood fill algorithm is used to count the number of pixels in the area. Individual characters are large objects and such can be easily distinguished from noise by empirically setting a certain threshold. The objects with area count under the threshold are then deleted, which results in an almost noiseless image.

3.2.2 Flood fill algorithm

The flood fill algorithm, also known as bucket fill or seed fill, is a simple algorithm that determines the area of an object with a certain color in an image matrix. It can be used in two ways. In the first one, the aim is to count the number of pixels in the object. In the second one, it is used to paint the object with a certain color. This property is useful for removing an object, because we can just set the replacement color to the background one.

3.3 Canny edge detection

3.3.1 Introduction

Edge detection is a standard problem in the field of image processing. It aims to identify the areas where the image brightness or color intensity changes sharply or in other words the boundaries of objects within the image. In our effort to break captchas, it can be successfully employed, for example to detect the edges of characters on a particularly confusing background. Moreover, some advanced features do not require the whole letter but only its outline, which can also be achieved with an edge detector. One of the best known algorithms is the Canny edge detector. This algorithm was presented by John Canny in one of his papers[12]. He analyses the problem and introduces the following criteria:

Algorithm 1: Flood fill algorithm

```
Data: Start node, target color, replacement color
Result: Pixel count
if target color is equal to start node color then
    return 0
end
Set $C$ to 0
Set $Q$ to empty queue
Add the start node to $Q$
while $Q$ not empty do
    Dequeue node $N$ from $Q$
    if color of $N$ is the target color then
        Set the color of $N$ to replacement color
        Increase $C$ by 1
        Enqueue nodes north, east, south and west from $N$ to $Q$
    end
end
return C
```

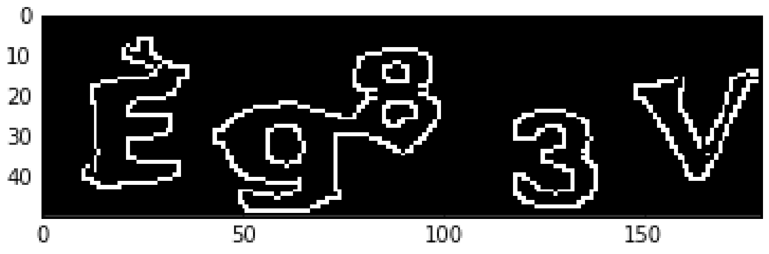


Figure 3.6: Effect of the Canny edge detection algorithm

1. **Low error rate:** It is vital that the error rate is low. All the edges should be found and the algorithm should produce no spurious responses.

2. **Localization:** The distance between the points found by the detector and the true edge center should be minimal.

3. **Minimal response:** There should be no other responses to a single edge but one.

3.3.2 Process

The algorithm consists of these steps:

1. **Noise reduction**

In order to fulfil the first criterion we need a way to eliminate noise. It is achieved by applying a Gaussian kernel filter to convolve with the image. The equation to calculate the kernel is:

$$G_{xy} = \frac{1}{2\pi\sigma^2}\exp(-\frac{(x-(k+1))^2+(y-(k+1))^2}{2\sigma^2}); 1 \le x, y \le (2k+1) \tag{3.10}$$

for a kernel size of $(2k+1)\times(2k+1)$. The equation for the new smoothed image, where A is the original image matrix and B is the result[3], is this:

$$B = G * A \tag{3.11}$$

It will slightly smooth the image reducing the effect of noise on the detector while preserving its ability to find edges.

2. **Finding intensity gradient**

   The smoothed image is then filtered with an edge detection operator. Canny himself did an extensive research in this aspect and found an optimal function, which is described by a sum of four exponential terms. However in order to effectively compute the two-dimensional extension of the filter, he found that a close approximation is the first derivative of a Gaussian $G'(x)$, where

$$G(x) = e^{-\frac{x^2}{2\sigma^2}} \tag{3.12}$$

   To this day, a number of edge detection operators have been proposed, each of them having different properties and results. A popular operator, which has also been chosen in this paper, is the Sobel operator[13]:

$$\mathbf{G}_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * \mathbf{A} \tag{3.13}$$

$$\mathbf{G}_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * \mathbf{A} \tag{3.14}$$

   which gives us the approximated first derivative in horizontal and vertical direction. Using this tool, we can easily calculate the edge gradient map G. The angle of each edge is also calculated by means of the arctangent function with two arguments.

[3]Note that the $*$ symbol denotes the convolution operation

$$\mathbf{G} = \sqrt{\mathbf{G}_x{}^2 + \mathbf{G}_y{}^2} \tag{3.15}$$

$$\boldsymbol{\Theta} = \operatorname{atan2}\left(\mathbf{G}_y, \mathbf{G}_x\right) \tag{3.16}$$

As we are working on a pixel matrix, the edge direction angles are rounded to the nearest vertical, horizontal or diagonal lines, which correspond to 0°, 90°, 45° and 135° respectively.

3. **Non-maximum suppression**

   Given the minimal response criterion, we apply an edge thinning technique called the non-maximum suppression. After applying the smoothing filter and calculating the intensity gradient, the edge is still somewhat blurred. All pixels which are not a true edge but are adjacent to it are therefore suppressed. It is done by comparing the pixel edge strength value in the gradient map to pixels which lay perpendicular to its direction and are facing the same direction, and suppressing all but a local maximum. For example a pixel P is an edge in vertical direction. It is compared with its neighbors on the horizontal line. If they have a lower value than P, it is the local maximum and they are suppressed.

4. **Hysteresis thresholding**

   So far we have had a close approximation of the image edges. Nevertheless, still a few of the edges are spurious responses caused by various noise. In order to eliminate them, we apply a double threshold. The first threshold value is higher and selects the sure edges which will be kept. The second threshold value is lower and eliminates non-edges which are below the lower threshold and are then deleted. The pixels whose values are between the two thresholds indicate weak edges which can be either real edges or just noise. To distinguish between them, we look at their position in the image. If they are connected to a sure edge, they are also considered a real edge, otherwise they are treated as noise and are thus deleted. The two threshold values are usually set by the user, as different images have different properties. They can also be determined using the Otsu's thresholding method already mentioned earlier in 3.1. The final image is a close representation of the image edges.

Chapter 4

Segmentation and post-segmentation

Due to the increasingly greater success rates of machine learning algorithms at recognizing individual characters, it has been generally acknowledged that captchas security lies in withstanding the segmentation. Most captchas have employed at least some form of anti-segmentation techniques. In my research various segmentation methods were studied and assessed and subsequently the following segmentation algorithm was designed.

In our case, we have been fairly successful in eliminating the noise from the images. With our simplistic approach, only the individual characters and a few lines remain. At first we isolate all the objects left in the image, which is done by iterating through every pixel. When an unlabelled pixel with a foreground color is found, the flood fill algorithm is used to paint it a new unique color, which also serves as a label. The preprocessing stage of the algorithm is not completely efficient, so some occasional lines remain in the image. They can be either connected to a character or isolated. Due to the nature of the lines used, their position is generally horizontal. That is unlike any of the characters the captchas contain and as such the isolated lines can be easily eliminated by deleting all objects with their height under certain empirically set threshold.

If the number of the acquired objects is the desired number of characters, the captcha is considered successfully segmented, and we move to the next step. In the other case, we have two possibilities. The first one is that the number of objects is greater than number of characters. This implies that there are some speckles or line segments left. They are eliminated by iterating through all the objects and deleting the ones with the lowest pixel count. This usually provides good results. The other possibility is that we have fewer objects than the number of characters which indicates a connection of multiple characters either by a remaining line or by collapsing. This situation is resolved with an x-axis projection algorithm.

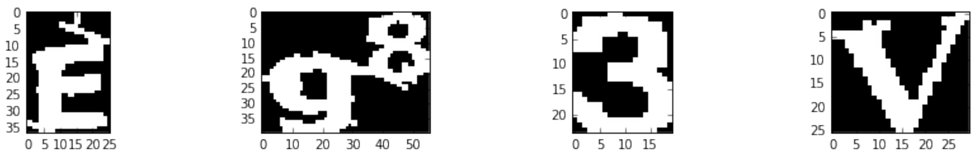


Figure 4.1: Partial segmentation with two connected characters

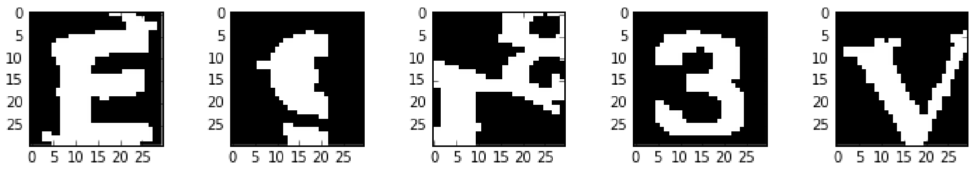


Figure 4.2: Segmentation attempt with the cut in the middle of a character

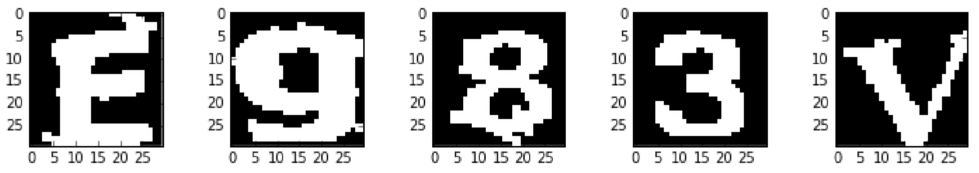


Figure 4.3: Correct segmentation with the cut between two characters

4.1 X-axis projection

The idea of X-axis projection is based on the notion that when two or more characters are joined, the pixel count between them is generally lower than in the centre of the character. First, we construct the X-axis projection by summing the pixels of each column. In order to reduce the effect of noise and small count fluctuation, a smoothing filter is applied to the projection. Next, all local minima are found which will be later considered for cutting points. Note that some adjacent columns can have the same pixel count and be all at the local minima. In that case we consider only the middle one to be a cutting point. The next step is to remove all local minima which have their pixel count under empirically set threshold to eliminate most cutting points positioned in the middle of a character. This gives us a number of possible ways to segment the joined characters. Note that more cutting points than desired are usually found. It is nevertheless not an issue, because we will retain all the possible segmentations, recognize them and select the one which maximizes the classification score.

4.2 Features

After successfully segmenting the image and obtaining the individual letters, they are further transformed to prepare for the classifier and to optimize its performance. It operates on vectors with the same length and so the data has to be stored in a matrix with fixed dimensions. Because every character has different size, we need decide how the data will be handled.

The most primitive approach is to use the raw data which are in other words the portion of the image matrix containing the character. This approach can be divided into two options: In the first one, the whole object is treated as the character and is simply resized to the desired proportions. It stretches the thin characters and shrinks the wide ones. The same characters are stretched equally so it does not impede the classifier performance. The second assumes the object contains some remaining noise. In order to reduce its effect on the performance, the center of mass is calculated and the center of the image is shifted to it. This reduces the effect of any remaining lines connected to the character.

These approaches have a huge disadvantage in containing an unnecessarily large amount of data, which impairs the computational capabilities of the classifier. It can be partially reduced by scaling the character down. Its main advantage lies in its simplicity and a sufficient resistance to noise. A further improvement can be made by using a set of weights preferring the center of the character. The idea behind this is that the center is less affected by the distorting noise than the outward regions which can for example be overlain by another character.

A more progressive approach is to extract advanced features from the raw data which accurately distinguish between the characters. They significantly reduce the amount of data we have obtained making it easier to work with. Their main advantage though lies in their properties, because depending on what features we extract, they can be invariant to translation, scale, rotation, skewing, stretching, mirroring, and so on. They also offer various resistance to noise.

4.2.1 Hu moments

The features of my choice are Hu moments. They are a set of invariants derived from image central moments which are particular weighted averages of the intensities of the images' pixels. Simple image properties can be derived from these moments, for example the area, the centroid or its orientation. These invariants are by construction invariant to translation and can be generalized to scale invariance. In the work of Hu et al[14], moments invariant to rotation were constructed conveniently counteracting the effect of character rotation in captcha designs. The central image moments invariant to translations for greyscale images with pixel intensities $I(x, y)$ are defined as follows:

$$\mu_{pq} = \sum_x \sum_y (x - \bar{x})^p (y - \bar{y})^q f(x, y) \tag{4.1}$$

These can be generalized to moments by normalization with scaled zero-th central moment. The resulting moments are invariant to translation and scaling.

$$\eta_{ij} = \frac{\mu_{ij}}{\mu_{00}^{\left(1+\frac{i+j}{2}\right)}} \tag{4.2}$$

where $i+j \geq 2$. In his work, Hu proposed moments invariant to translation, scale, and additionally to rotation. They can be constructed as follows:

$$I_1 = \eta_{20} + \eta_{02} \tag{4.3}$$

$$I_2 = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2 \tag{4.4}$$

$$I_3 = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2 \tag{4.5}$$

$$I_4 = (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2 \tag{4.6}$$

$$\begin{aligned} I_5 = {} & (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] \\ & + (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] \end{aligned} \tag{4.7}$$

$$\begin{aligned} I_6 = {} & (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] \\ & + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03}) \end{aligned} \tag{4.8}$$

$$\begin{aligned} I_7 = {} & (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] \\ & - (\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] \end{aligned} \tag{4.9}$$

Chapter 5

Recognition

Recognition is the last part of the captcha attack pipeline. It works in two modes. In the learning mode, the algorithm is told what the deciphered captcha text is and it uses the information to learn what each letter looks like and stores it in its database. In the testing mode, this stage is used to recognize each segmented letter and produce the most probable answer.

There are three most frequently used classifiers for breaking captcha. The first one is a SVM classifier which stands for “Support Vector Machine”. In short, it is a supervised learning algorithm which constructs an optimal hyperplane boundary in an n-dimensional space between two classes. In addition to linear classification, it can be generalized to non-linear classification with the use of a kernel. Its disadvantage is being prone to over-fitting.

The next common classifier family is an Artificial Neural Network or ANN for short. It is a computational model inspired by biological neural networks with layers of interconnected neurons. Each neuron processes its input with a weighted function and sends the output signal to the next neuron layer. It is a non-parametric model and as such does not need a statistical background to set the right parameters. Another advantage is its ability to learn detecting all possible interactions between the variables. Its disadvantages are that it is a “black box”, we do not know the inner workings of the network and cannot use it to our advantage. It is also computationally demanding and complicated to implement. For these reasons, it is not the algorithm of my choice. For more information on both classifiers please refer to [15].

5.1 K-nearest neighbors

The last classifier and the classifier of my choice is the K-nearest neighbors. It is a simple algorithm with only a few parameters, but with good results. Unlike the SVM algorithm, the KNN permits discontinuous classes, which is very useful in classifying multiple fonts and letter weights into single class, making it a very variable algorithm. It is an example of the so-called “lazylearning" algorithm where in the training mode all examples are only stored in a database and all the computing is done in the testing phase.

Now I will give a short summary of how the algorithm works. First we need to learn the classifier on a training set which consists of labelling and storing the feature vectors in a database. Then, in order to classify a given point, we compute how far it is from each of the learned examples and choose k nearest neighbors. The distance can be calculated with one of numerous metrics. They will be discussed later in this work. In the last step, a label is assigned to the given point based on the majority vote of its neighbors. It can also be chosen based on a weighted vote, where the weight w is usually determined as $w = \frac{1}{d}$ with d as the distance from the given point.

This classifier offers a few parameters to optimize the performance with. The most prominent is the choice of k or, in other words, how many nearest neighbors will compete in the final vote. Low number of neighbors may lead to "overfitting", that is a state where the model is highly sensitive to noise and random error. The classifier then chooses the label based on only the few nearest neighbors which do not have to necessarily correspond with the majority of the class. High number of neighbors will however result in "underfitting" where the model generalizes the underlying relationships too much and results in poor classifier performance. Next area of optimization is the decision between the majority vote and the weighted vote. It results in only a moderate difference between the outcomes, but can be easily used in the fine tuning of the algorithm.

5.2 Metrics

The last choice is between different distance metrics. There is an extensive number of ways to compute the distance so I will mention only the most common ones. A distance function or a metric is a function that defines a distance between each vector within a set. They play a vital role in machine learning and in classification, because they provide us a tool to measure how similar or distant two vectors are. A function $d : X \times X \to \mathbb{R}$ on a set X is a metric if following conditions are satisfied:

$$d(x,y) \geq 0 \qquad \textit{(non-negativity)} \tag{5.1}$$
$$d(x,y) = 0 \text{ if and only if } x = y \qquad \textit{(identity of indiscernibles)} \tag{5.2}$$
$$d(x,y) = d(y,x) \qquad \textit{(symmetry)} \tag{5.3}$$
$$d(x,y) \leq d(x,z) + d(z,y) \qquad \textit{(triangle inequality)} \tag{5.4}$$

5.2.1 Euclidean metric

Also called the Pythagorean distance, it is the most common and one of the most basic distances. In Euclidean space, it is the length of a line connecting to points. Let d_E be the Euclidean metric between two vectors on the $\mathbb{R}^n$ space. It is defined by[16, p. 94]

$$d_E = \|x - y\|_2 = \sqrt[2]{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{5.5}$$

5.2.2 Manhattan distance

In the case where we use the distance to compare the results of k nearest neighbors in the majority vote in the last step of the KNN algorithm, the square and the square root operations can be removed with the same result. The outcome is the city block distance named after the grid layout of the streets of Manhattan where it is the shortest distance a taxicab drive take between two intersections. The optimized equation results in[16, p. 323]:

$$d_M(x, y) = \|x - y\|_1 = \sum_{i=1}^{n} |x_i - y_i| \tag{5.6}$$

Note that because we are not working on real numbers but only on binary vectors $x, y \in \{1, 0\}$, the distance is equal to the Hamming distance which is, in other words, the total number of different bits of two binary vectors.

5.2.3 Cosine distance

A cosine distance is a dissimilarity measure of two vectors measuring the cosine of the angle between them. The result is 0 for vectors with the same orientation and 1 for angles perpendicular to each other. As such it does not measure the magnitude of the factors but rather the orientation.

$$d_C(x, y) = 1 - \frac{\sum_{i=1}^{n} x_i * y_i}{\sqrt{\sum_{i=1}^{n} x_i^2} * \sqrt{\sum_{i=1}^{n} y_i^2}} \tag{5.7}$$

5.2.4 Jaccard distance

The Jaccard distance is again a dissimilarity measure of two vectors. In some cases the presence of an object does not hold the same weight as its absence. Therefore the Jaccard distance takes into account only the objects which are present in at least one of the sets.

$$d_J(A, B) = 1 - \frac{A \cap B}{A \cup B} \tag{5.8}$$

Chapter 6

Experimentation

In this chapter, the implemented algorithm is tested on real captcha images in multiple experiments. Various parameters are tried and evaluated.

As previously stated, a captcha is considered compromised, if at least 1% of attacks are successful. Accuracy is nevertheless not the only criterion. In order to be effective, the algorithm has to perform in sufficiently short time. A solution in which a captcha can be attacked 10 times per second, but has only a 20% recognition rate, is more effective than a solution with recognition rate of 90% with attacks only once a second.

6.1 Data

Before the algorithm can be used to break the captcha, the KNN classifier has to be learned first on a training dataset. To train the classifier, a large number of captcha images has to be acquired and manually labelled. It can be quite a challenge, because the more difficult the captcha is, the lower is the human recognition rate. The resulting errors in the labelled data lead to an incorrectly learned classifier and a reduced recognition accuracy. The recognition rate is also directly dependent on the data amount, but acquiring a sufficient volume can be quite time consuming, which can deter some of the attackers. Also to test the recognition rate correctly, the classifier must learn on a different dataset than the testing set. This fact further increases the amount of data that had to be acquired.

6.1.1 ČÚZK data

Fortunately for the attacker, the captcha on cuzk has a serious design flaw. The captcha shown on the image is not a standard GIF or JPEG format but rather a ".axd" file, which is a HTTP Handler file used by ASP.NET applications and is generated on runtime. Simply refreshing the image (not the whole page) then generates a new captcha challenge containing the same characters.

This flaw allows us to write a simple script which refreshes and downloads the image numerous times in a row. Labelling the data is then significantly easier considering only one character sequence for the whole set of images has to be recognized. It also makes it much more accurate, because if any of the characters in a captcha is for some reason illegible, that image can be skipped and the characters can be recognized based on another legible image.

For the following experiments, 21 sets of captchas were downloaded and labelled for learning purposes. Each set contains 100 different images. The resulting training dataset contains approximately 10, 500 characters. Representation of characters in the dataset is relatively uniform and so an average of roughly 170 examples per character is acquired. The experiments were conducted on the whole set or on a subset. The independent testing dataset contains 100 unique manually labelled captcha images.

6.1.2 mojedatovaschranka data

This website does not contain any similar security flaw and accordingly all data had to be labelled manually. A smaller training dataset of only 25 images with 125 individual character examples was thus obtained. However, as this scheme contains only numerics and virtually no noise, a set of average 12.5 unique examples per character is sufficient. The testing dataset contained 50 images.

6.1.3 ulozto data

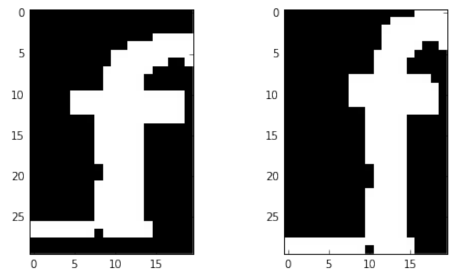
A dataset containing 50 images was stored and manually labelled. It was later used as a testing dataset with a training dataset from previous experiments. Therefore no training database of its own was required.

6.2 Experiments

This section summarizes the experiments that have been conducted.

6.2.1 Pre-processing optimization

Optimizing the parameters of the pre-processing stage can have a great effect on the algorithm efficiency. First we compare whether it is better to stretch the characters to fit in the bounding box, in which they will be stored, or to leave them in their original scale. When the scale is preserved, the characters are centered on their center of weight to increase the performance. Examples can be seen in Fig 6.1.



(a) Centred data (b) Resized data

Figure 6.1: An example of different storing modes with the character *f*

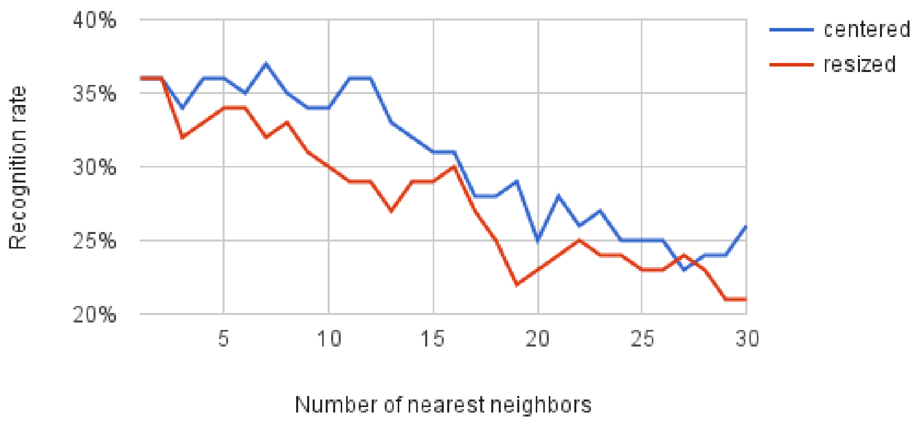


Figure 6.2: Recognition rates based on the data storage mode

As we can see from the graph in Fig 6.2, the two modes achieved similar results. The centered mode performed slightly better peaking at 38%, which can be attributed to better resistance to noise. Both were trained on a dataset containing 4, 120 characters. The average time to solve a captcha was 0.75 seconds.

Next we analyze whether the size of the stored characters impacts the classifier performance. Three different sizes were tested of 20×20, 30×30 and 40×40 pixels per character. The results in graph in Fig 6.3 suggest that the smaller size decreases recognition rates only slightly. However, there was an increase of 16% in average runtime, which suggests that storing smaller samples is overall beneficial.

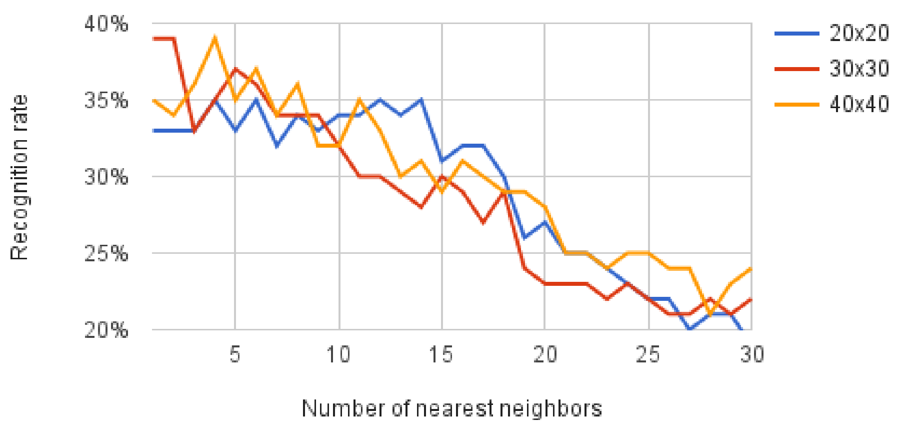


Figure 6.3: Recognition rates based on the data storage size

6.2.2 Hu moments

In this experiment, we test whether Hu moments are going to perform well enough and produce sufficient results. A training dataset was created containing again 4, 120 characters. Recognition of each of the 100 images from the testing dataset was attempted, but it was a total failure with 0% success rate. So as to find out if at least some of the characters can be recognized, the dataset was normalized to the range $(0, 1)$ to eliminate the effect of different scale of attributes and then cross-validated. The resulting recognition rate was 13.86%. The confusion matrix can be seen in Fig 6.4. The low accuracy results presumably from the low tolerance to noise and the choice of KNN algorithm. The main motivation behing Hu moments to reduce the amount of data and lower the calculation time. Nevertheless, the cost of computing the features is far higher than the cost of calculating the distances on smaller vectors leading to up to 55% increase in computational time.

6.2.3 Classification optimization

The last area of optimization is the recognition stage. Numerous distance metrics are analysed and the effect on various number of nearest neighbors in the KNN algorithm is evaluated. The learning dataset size was increased to 8, 315 entries and the testing dataset is the standard 100 images. The results peaking at 46% can be seen in Fig 6.5. The bigger learning dataset had increased the recognition rates by the average of 5% without hardly any deceleration. We can see that all the metrics produce similar results with the Manhattan distance achieving the highest recognition rate. Moreover, Manhattan distance was calculated in the shortest span of time seen in Table 6.1 making it the best distance to use. The results refuted the idea that

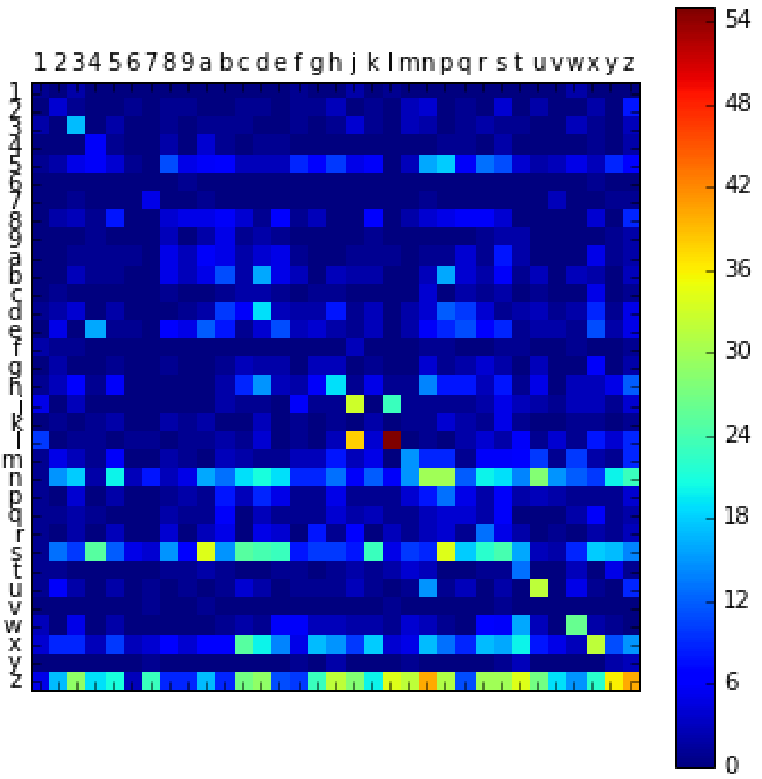


Figure 6.4: Confusion matrix using Hu moments. On the vertical resp. horizontal axis is the predicted resp. true value

Distances	Manhattan	Cosine	Jaccard	Weighted Manhattan
Total time	106 s	190 s	158 s	202 s
Time relative to fastest	1.00	1.79	1.49	1.91

Table 6.1: Table of calculation times depending on metric type

weighing the features prioritizing the inner region of the character will increase the recognition rate.

The performance of the dataset containing 8, 315 entries has been assessed in a tenfold cross-validation test. This means ten experiments were run each having one tenth of the data as learning data and the rest as testing data. We were able to achieve an 85.5% accuracy. We were not able to achieve results comparable to modern optimized classifiers, but they are more than sufficient for our algorithm. The confusion matrix produced can be seen in Fig 6.6. We can clearly see that similar characters are often mistaken with each other.

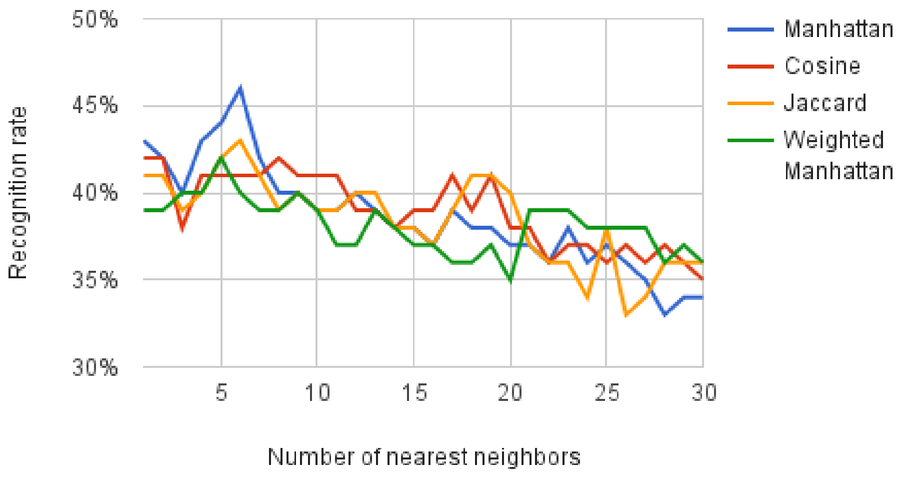


Figure 6.5: Graph of recognition rates of different distance metrics depending on the number of nearest neighbors

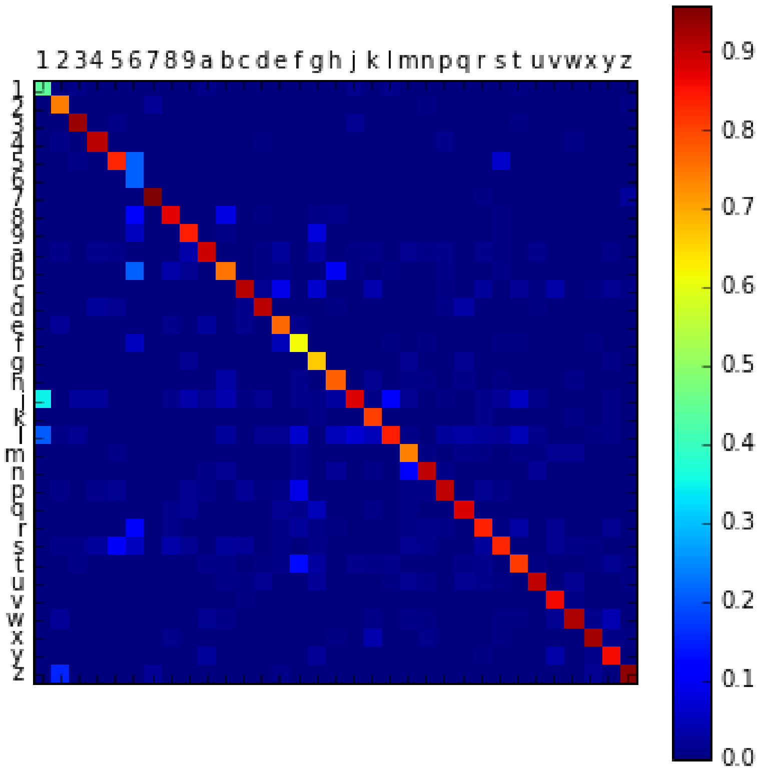


Figure 6.6: Confusion matrix of training dataset.

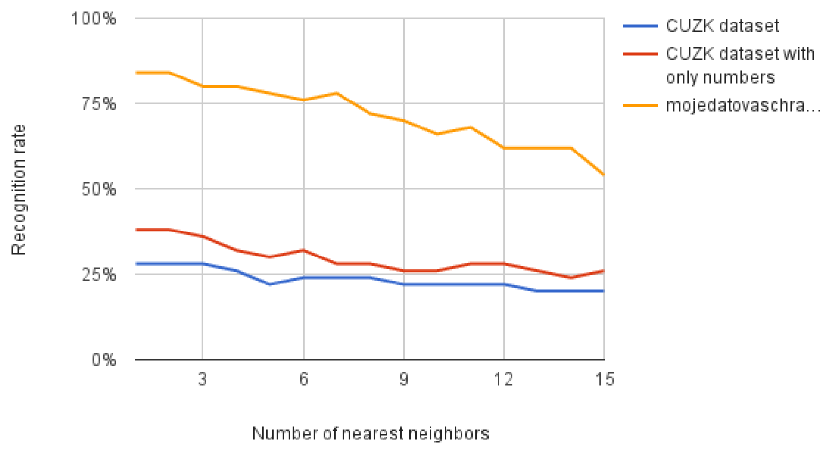


Figure 6.7: Graph of different dataset accuracy rates

6.2.4 Application on different solution

These experiments will be dealing with using the previously trained algorithm on different captcha solutions. The first will be the ulozto captcha with 50 images in the testing dataset. When the dataset of 4, 120 entries of 20x20 pixels with the Jaccard distance was used, the algorithm segmented and recognized up to 14% of the images. Even though the accuracy is pretty low, it is still far above the 1% threshold for the captcha to be compromised. Bearing in mind that the classifier was not trained on the attacked captcha but on a different one, it is actually quite a big success.

Next, the trained algorithm was deployed on the mojedatovaschranka captcha. After optimizing the parameters, the resulting recognition rate was up to 28% with the original dataset containing both numbers and letters. This captcha solution does not contain any letters so they can be removed from the dataset. Eliminating confusion of *g* with *9* or *l* with *1* etc. increased the recognition rate to 38%. The reduction of size had the benefit of up to three times faster calculation speed.

To further examine the dataset influence on the classifier accuracy, an independent training dataset for the mojedatovaschranka captcha was created with 25 images containing total of 125 entries. It may seem to be a rather small dataset, but as the solution contains only 10 numerals, the average is 12.5 entries per character, which turns out to be sufficient. The Jaccard distance metric yielded the best results with an impressive 82% recognition rate. The graph containing the comparison between different training datasets can be seen in Fig 6.7.

Conclusion

This thesis aimed to evaluate the effectiveness of various feature designs in text-based captcha schemes. The main goal was to implement an algorithm capable of recognizing characters in various solutions.

First, numerous anti-recognition and anti-segmentation features were described, and their impact was evaluated. Then, an algorithm for character recognition in captcha solutions was presented. It involved a custom segmentation algorithm capable of isolating the individual characters based on density of pixels in the image.

The proposed algorithm was successfully implemented and achieved up to 46% recognition rate on the cuzk solution and 82% on the mojedatovaschranka solution. A recognition rate of 14% was achieved on the ulozto solution, even though the algorithm was trained on a different solution, proving to a certain degree that the algorithm has a general application.

The algorithm can be further developed by improving the pre-processing stage to preserve all the character features while removing more noise. Also a different classifier, such as a neural network, could be used to achieve better recognition rates.

Bibliography

[1] Von Ahn, L.; Blum, M.; Hopper, N. J.; et al. CAPTCHA: Using hard AI problems for security. In *Advances in CryptologyEUROCRYPT 2003*, Springer, 2003, pp. 294–311.

[2] Turing, A. M. Computing machinery and intelligence. *Mind*, volume 59, no. 236, 1950: pp. 433–460.

[3] Chellapilla, K.; Larson, K.; Simard, P.; et al. Designing human friendly human interaction proofs (HIPs). In *Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM, 2005, pp. 711–720.

[4] Bursztein, E.; Martin, M.; Mitchell, J. Text-based CAPTCHA strengths and weaknesses. In *Proceedings of the 18th ACM conference on Computer and communications security*, ACM, 2011, pp. 125–138.

[5] Yan, J.; El Ahmad, A. S. CAPTCHA security: a case study. *IEEE Security & Privacy*, , no. 4, 2009: pp. 22–28.

[6] Nahlížení do katastru nemovitostí [online]. 2004-2016, [Cited 2016-06-06]. Available from: `http://nahlizenidokn.cuzk.cz/`

[7] Datové schránky [online]. 2016, [Cited 2016-06-06]. Available from: `https://www.mojedatovaschranka.cz/`

[8] Ulož.to [online]. 2016, [Cited 2016-06-06]. Available from: `http://ulozto.cz/`

[9] OpenCV 3.1.1 Documentation [online]. December 2015, [Cited 2016-23-04]. Available from: `http://docs.opencv.org/3.1.0/de/d25/imgproc_color_conversions.html`

[10] BT.601:Studio encoding parameters of digital television for standard 4:3 and wide screen 16:9 aspect ratios [online]. October 2015, [Cited 2016-06-05]. Available from: `http://www.itu.int/rec/R-REC-BT.601/`

[11] Otsu, N. A threshold selection method from gray-level histograms. *Automatica*, volume 11, no. 285-296, 1975: pp. 23–27.

[12] Canny, J. A computational approach to edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, , no. 6, 1986: pp. 679–698.

[13] Sobel, I.; Duda, R.; Hart, P.; et al. History and Definition of the Sobel-Feldman Operator. 2014.

[14] Hu, M.-K. Visual pattern recognition by moment invariants. *information Theory, IRE Transactions on*, volume 8, no. 2, 1962: pp. 179–187.

[15] Cortes, C.; Vapnik, V. Support-vector networks. *Machine learning*, volume 20, no. 3, 1995: pp. 273–297.

[16] Deza, M. M.; Deza, E. Encyclopedia of distances. *Encyclopedia of Distances*, 2009: pp. 1–583.

Appendix A

Acronyms

CAPTCHA	Completely Automated Public Turing test to tell Computers and Humans Apart
SVM	Support Vector Machine
KNN	k-Nearest Neighbors
ANN	Artificial Neural Network
RGB	Red Green Blue
JPEG	Joint Photographic Experts Group
GIF	Graphic Interchange Format
HTTP	Hypertext Transfer Protocol

Appendix B

Contents of enclosed CD

```
readme.txt ........................ the file with CD contents description
src ...................................... the directory of source codes
  impl ....................................... implementation sources
  thesis .............. the directory of LaTeX source codes of the thesis
text ......................................... the thesis text directory
  BP_Pistora_Matous_2016.pdf ......... the thesis text in PDF format
```