Optical Character Recognition of Heavily Distorted Text Segments

a Case Study on reCaptcha Security

Master's Thesis

in Computer Science and Engineering

submitted to the
Institute of Software, Technology & Systems
of the
Hamburg University of Technology

in Partial Fulfillment of the Requirements for the Degree of
Master of Science

by
Alexander Motzek

Advisors

Prof. Dr. rer. nat. habil. Ralf Moeller
Prof. Dr.-Ing. Rolf-Rainer Grigat

Hamburg, January 2014

Abstract

The fact that artificial text segments can be generated, which are recognizable by humans but not by artificial intelligence, shows that research in the field of artificial intelligence in the range of optical character recognition is not as advanced as it could be. Such artificial text segments serve as human interaction proofs in the field of computer security mechanisms, also known as *CAPTCHA*s.

Taking one of the most used CAPTCHAs—the reCaptcha system—as a guideline for heavily distorted text segments, this thesis pursues the goal of developing a robust recognition algorithm, which is easily adaptable to other systems and use cases.

In the scope of this thesis a strong character classifier based on *Fourier descriptors* is developed. The test results indicate that the proposed algorithm outperforms current state-of-the-art character recognition systems in single character recognition with a mean recognition rate of 79% vs. 39%. Furthermore, with a recognition rate of 29% it is able to break the reCaptcha system and beat human recognition performance by nearly 50%. On top of that, it is shown to be adaptable to other CAPTCHA systems and, despite being developed for character recognition, also to numbers.

The complete algorithm solely depends on pre-rendered instances of characters without any knowledge or instances of applied distortions and is able to recognize and separate merged characters.

Contents

1 Introduction and Motivation **3**

2 CAPTCHA **5**

2.1 History and Overview of Captcha Systems and Related Work 5

2.2 The reCaptcha System 8

2.3 Model Development and Data Acquisition 8

2.3.1 Sloping 9

2.3.2 Slanting 9

2.3.3 Connection of Characters 10

3 Word Processing **11**

3.1 Separation of Pro-Bono and Challenge 11

3.2 De-Sloping 12

3.3 De-Slanting 14

3.4 Evaluation 16

4 (Over)Segmentation **18**

4.1 Separated Segments 18

4.1.1 Broken, Over- and Under-Connected Characters 19

4.2 Oversegmentation 20

5 Character Classification **23**

5.1 First Ideas 23

5.2 Feature Extraction 24

5.2.1 Character Shape 24

5.2.2 Feature Removal and Detection of i-Dots 27

5.2.3 Necessary Condition – Height and Minimal Area 28

5.2.4 Necessary Condition – Open Counters 29

5.2.5 Necessary Condition – Stems 30

5.2.6 Necessary Condition – Holes 30

5.2.7 Classification Engine 31

5.3 Evaluation . . . 33
5.3.1 Self-Distorted . . . 33
5.3.2 reCaptcha Characters . . . 34

6 Backtracking and Evaluation **37**
6.1 Viterbi . . . 37
6.2 Further Tweaks . . . 37
6.3 Evaluation . . . 39
6.3.1 String Difference Measure . . . 39
6.3.2 Human Performance . . . 40
6.3.3 Evaluation of Tesseract and Finereader, unprocessed . . . 40
6.3.4 Evaluation of Tesseract and Finereader, processed . . . 41
6.3.5 Evaluation of the Proposed Algorithm . . . 42
6.3.6 Live Test / Verification Set . . . 44

7 Outlook and Conclusion **47**
7.1 Postprocessing . . . 47
7.2 Adaptability . . . 49
7.3 Conclusion . . . 51

A Fourier Descriptors **53**
A.1 Mathematical Description and Generation of Fourier Descriptors . . . 53
A.1.1 Generation of Fourier Descriptors . . . 53
A.1.2 Invariances . . . 54
A.1.3 Effects of Mirroring and Further Derivations . . . 55
A.2 Normalisation . . . 58

B Confusion Matrices **59**

Bibliography **67**

List of Figures **69**

List of Tables **71**

Chapter 1

Introduction and Motivation

This thesis will deal with the problem of optical character recognition on heavily distorted text segments, likely to be found e.g. in historic books or low resolution digital copies of prints.

While it seems that research in classic optical character recognition of Latin characters has been abandoned since the late 90s, the fact that still text segments can either be artificially or naturally generated unrecognizable by modern OCR techniques shows that the research is not as complete as it could be. Modern OCR systems work extremely well on high resolution scans but rapidly fail on low quality.

As part of security mechanisms so called captchas still rely on the generation of text segments that are recognizable by humans but not by computers. That this is possible, shows how little these systems are advanced in dealing with non-standard inputs.

On the other hand the fact that these captchas are needed as a security mechanism shows that there must be a demand in breaking them, i.e. there is a demand for new research in this area.

In this thesis we will first identify a model for common types of distortions and then further on try to develop an algorithm allowing the digitalization of even heavily distorted text segments.

To do so we will conquer the challenge of breaking one of the most used captchas in the wild—the reCaptcha system under development of Google. With over 200,000 sites using reCaptcha [reC, `/whyrecaptcha`] it represents a striking attack target and at the same time an excellent model for heavily distorted text segments.

Yet, we only want to take these as guidelines in our research, just as very extreme examples of heavily distorted text segments. We want to focus on developing a robust algorithm, which is not tightly tailored to one particular system, but being easily adaptable with a very high generalization ability. We want to point out that breaking the system is not the main goal of thesis, but still is a very nice challenge to conquer. Quoting Luis von Ahn—one of the inventor of captchas and the reCaptcha system:

> A CAPTCHA implies a win-win situation: either the CAPTCHA is not broken and there is a way to differentiate humans from computers, or the CAPTCHA is broken and a useful AI problem is solved.
>
> – *Luis von Ahn, et al. [vBL04]*

It describes the balancing on a knife's edge in this thesis and its goals.

The rest of this thesis will be structured as follows: We will first give a brief insight into current captcha systems, their applications and history in the second chapter. Further on, we will start developing a model of heavily distorted text segments derived from instances of the reCaptcha system. In addition we will also try to find real world occurrences for legitimating the identified distortions.

From the third chapter on we will elaborate the algorithm. We will start with the processing of complete segments, in which we will deal with binarization, word-segmentation, desloping and deslanting. While considering the captcha as a complete word in the third chapter, the fifth chapter will deal with single characters and their classification. We will develop a very robust scheme to classify heavily distorted characters based on heavy feature extraction using Fourier descriptors.

The fourth chapter will build the bridge between the third and fifth chapter by building a segmentation algorithm that will separate the complete word into single characters. The idea will be to heavily over segment the word, from which the correct segmentation points will be derived by considering the confidence of recognition of consecutively merged segments.

The sixth chapter will bring the previous work together and finally propose the complete algorithm for word recognition. Finally, we will evaluate our algorithm and compare it to current state-of-the-art systems. We will conclude this thesis by giving an outlook to further possibilities and showing its high adaptability to other applications.

While developing the algorithm we will encounter many problems also common in the range of recognition of handwriting and shape recognition.

To the best of my knowledge the proposed algorithm features a new approach to optical character recognition and captcha security by resembling the way humans actually read and recognize characters and words and utilizes a new, heavy invariant character classifier based on Fourier descriptors and necessary-necessity-matching. In contrast to other works, the classifier is solely based on pre-rendered instances of characters from well known font faces and is not based on per-pixel accuracy or system-specific anomalies of certain characters. By heavy feature extraction from characters resembling the way humans are able to distinguish between different characters, it shows to be heavily adaptable.

I will speak of “we” in the sense of “The reader and I”, as I will take the reader by the hand and guide them through this thesis. The reader shall understand all assumptions and made decisions. We will try to break down all explanations to a reasonable level, neither requiring an in-depth knowledge of image processing, data mining nor optical character recognition.

Chapter 2

CAPTCHA

2.1 History and Overview of Captcha Systems and Related Work

We will start this section with a quote from Luis von Ahn, which puts the effects and need of captcha systems in a nutshell. Luis von Ahn is one of the persons, who invented captchas in 2000 and is also one of the founders of the discussed reCaptcha system:

> This week Time Magazine announced the results of their Time 100, a listing of the most influential people over the last year. In the past this list has included Barack Obama, Steve Jobs, and Al Gore. This year's winner: "moot," the creator of an underground bulletin board called 4chan. Not many people had heard of moot before, so it's no surprise to say that that the poll was completely manipulated by members of the site he created. At first, the poll had almost no protection against abuse, so members of 4chan wrote a program to vote for moot millions of times :)
>
> Although Time Magazine eventually decided to implement reCAPTCHA on their poll, it was unfortunately too late. Before Time added reCAPTCHA, members of 4chan had written a program that was able to submit over 20 million votes. After Time added reCAPTCHA, the ballot-stuffing program completely stopped working, and the members of 4chan were forced to spend thousands of human hours typing reCAPTCHAs by hand. Through all this effort (some of them singlehandedly spent 40+ hours per week typing CAPTCHAs), they were only able to submit about 200,000 more votes after Time implemented reCAPTCHA. 200,000 votes is a small number compared to the number of votes other candidates got.
>
> Use reCAPTCHA from the start on your polls and you will significantly raise the bar for spammers to be effective.
>
> – *Luis von Ahn [vA09]*

CAPTCHA[1] is an acronym for "Completely Automated Public Touring-Test (to tell) Computers (and) Humans Apart". As part of security mechanisms in order to prevent massive automated requests to services, they require the user to proof being human and not a robot. They are especially used to prevent mass-mailing (spam), mass-posting (comment spam), mass-voting (ballout-stuffing), mass-registration or mass-downloading. For a more detailed description and history one can consider von Ahns original paper [ABHL03] or many other papers like [BMM11, Kal, YE08]. For fairness reasons we shall notice that von Ahn was not the first to invent the idea of captchas, but rather coined this term. Also noticeable work in the field of "human interaction proofs" was made by Microsoft's researcher Kumar Chellapillah [CLSC05b, CLSC05a].

[1] due to typographical reasons we will write the acronym in lowercase

In the history of designing and developing such captcha systems, of course, also the other side of the coin was researched—the breaking of the captcha systems. Many papers on this topic exist providing many approaches. Noticeable are for example [YE09], featuring a character classification engine solely based on the number of pixels or [GWF12], an attack on the Yahoo! captcha system. With the Yahoo! captcha system being similar to the here discussed reCaptcha system, it still is different from it. While the Yahoo! captcha features both distorted characters and merged characters, the characters still remain rather sharp and easy to recognize [GWF12]. Captchas with arbitrary noise speckles and overlayed lines have already been cracked, for example by [HLBO10]. The reCaptcha system features heavily distorted characters, on which classic classification engines fail, and connected characters, while not adding artificial and unusual distortions or noise. The high security of reCaptcha also becomes clear from an attack developed by Burzstein [BMM11, Bur12]. Burzstein demonstrated a successful attack on various common captcha systems at the CSA in 2011, breaking almost all captcha systems in the wild [BMM11]. Nevertheless, it exactly failed on the reCaptcha system.

Speaking of attacks to (previous generations of) the reCaptcha system one should note [BBFM11], an unusual attack to a captcha system, not breaking it down to single characters, but approaching it as a complete solid word. It used a dictionary of 20,000 pre-rendered words, which shapes where then matched to the captcha. A very minimalistic attack to a slightly more recent generation of the reCaptcha system was proposed in [Weg10]. Currently Gao et al. try to adapt their proposed algorithm for the Yahoo! captcha to the current generation of the reCaptcha system [GWF+13].

For further reading one can consider the "PWNtcha" project [Hoc], showing various weaknesses and examples of previous captcha systems.

As more and more systems are broken, also new ideas in human interaction proofs are emerging. We will now give a brief three notable examples and—as classic text-only captchas are well covered in the literature—will also focus on different approaches.

The first is a geometrical puzzle implemented by "relink.us". Figure 2.1 shows some examples of this interaction proof. The human is required to click inside of the opened circle. While this is very easy for a human, a robot might struggle with it due to overlapping circles. Nevertheless, all circles are of a different colour and the target circle always features a bolder stroke width. Two observations that might easily break this captcha.

FIGURE 2.1: Different relink.us-captchas. The human has to click inside of the opened circle. Source: `http://www.relink.us/`

Next is a captcha implemented by "share-links.biz" as seen in Figure 2.2. A matrix consisting of coordinates and a target index embedded in the background. This captcha features many security mechanisms, which need to be overcome in order to break it. First the background index must be extracted from a noisy background pattern. Further, all coordinates (extractable by the html image map) have to be analysed and finally the correct position has to be submitted. On top of that, some indices are left out, disabling a simple replay-attack. Still, it should be fairly easy to automate. As the backgroundpattern is always the same, it can simply be erased by a difference with it. Further, only the background index, which consists of clean stripes after background-removal, has to be extracted. Fixing these stripes should be possible by a median-filter. Finally, only the extracted indices have to be classified. As their locations are known and a constant font is used this should be possible by a pixel-count attack. The reason why this system is not already broken might be the fact that it is only used on one single site and therefore does not represent an interesting target.

Last we will look at a very notable and emerging captcha system. Just as reCaptcha it features an API making it easy to implement and reuse. It is therefore wide spread (e.g. it is used by Western

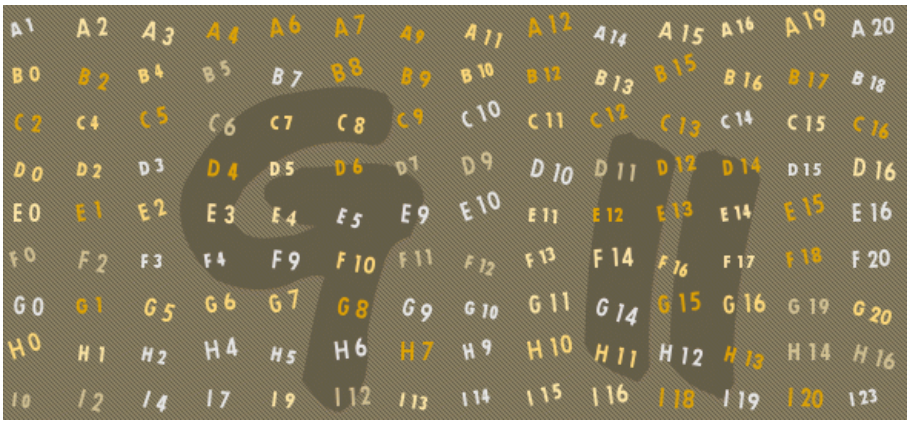


FIGURE 2.2: A "share-links.biz" captcha. The human has to click the denoted coordinate from the background. Notice that not all coordinates exist. Source: `http://share-links.biz/`

Digital and rapidgator) and forms an interesting target. It combines a human interaction proof with advertising and brand imaging. What makes it interesting is that it can be used in two ways. In an "advertising"-mode (Figure 2.3(bottom)) it features linguistic-challenging brand advertisements, requiring the user to enter a slogan or describe a brand. On the other hand it can be used in a "secure"-mode, featuring a classic text-based captcha which adapts to the input. It starts with a rather easy variant (Figure 2.3(top-left)), but is able to harden itself (Figure 2.3(top-right)) upon entering too many wrong challenges or when requiring a high reliability. On top of that, it does not use random words, but rather popular "geek"-phrases. With this even texts in extremely harsh conditions can be read by a human. Still, this also enables dictionary-based attacks.

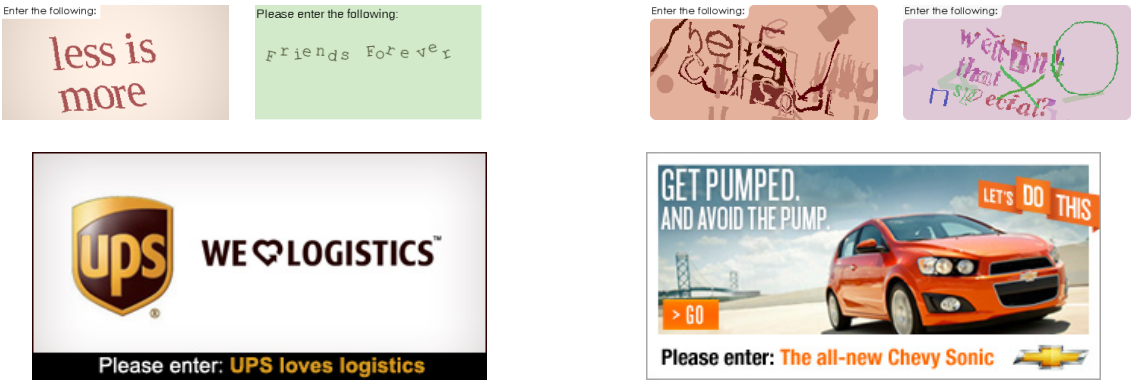


FIGURE 2.3: Different solveMedia captchas. It features a dynamic degree of difficulty (first row). The examples on the left are easy to solve compared to the ones on the right. Also, it features advertisement-based captchas requiring linguistic intelligence (second row).[2]

With this brief insight into current captcha systems and related works, we will now look into the current generation of the reCaptcha system on which we will focus in this thesis.

[2]Source: `http://www.solvemedia.com/` and their corresponding API.

2.2 The reCaptcha System

The reCaptcha system was invented in 2008 by Luis von Ahn, Ben Maurer, Colin McMillen, David Abraham and Manuel Blum. It started out as a project at the Carnegie Mellon University and was acquired by Google in September 2009 [vAC09, AMM+08][reC, `/aboutus`].

The system serves two purposes. On the one hand it provides security as a captcha system, on the other hand it helps digitizing books.

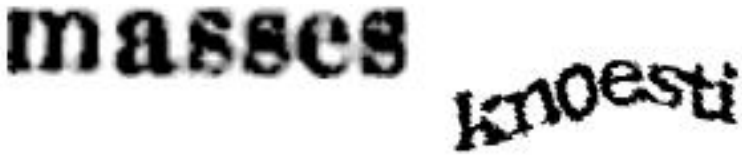


FIGURE 2.4: The Google reCaptcha[3]. Consisting of pro-bono part "masses" at the left and the challenge part "knoesti" at the right.

A reCaptcha always consists of two parts. One part being from the real world that was (not entirely) recognizable and one artificially generated text segment.

We will call the real world example the "pro-bono"—parts extracted from the Google Books project and from old New York Times journals [reC, `/learnmore`]. These segments are often very easy to read, but seem to confuse some OCR system or get recognized with low confidence. By asking the users for this text, Google can take a majority vote and improve their systems. For an in-depth and detailed description and analysis of this, one can refer to the original reCaptcha-science paper from von Ahn in [AMM+08].

The second part will be called the "challenge". It is the artificially generated text segment we are interested in. It was especially designed to look like a real world text segment while still being unrecognisable by computers. As it is not artificial and unusually scrambled with lines or artefacts, it reflects common real world distortions. We will construct the algorithm based on observations made from those captchas in order to hopefully provide new research results in the field of digitalizing heavily distorted (old) prints or text segments. As it resembles real world distortions, the research on this captcha system should be generously applicable to real world text segments.

Interestingly only the challenge part of the captcha must be entered to pass the test. Yet, entering something completely arbitrary for the pro-bono appears to let the test fail. This might be connected to a prank played on Google a few years ago where a group of people would always just type in the same word for this pro-bono part in the hope for it appearing in digitalized books [Red].

2.3 Model Development and Data Acquisition

In order to develop a model for heavily distorted text segments based on observations from the reCaptcha challenges, a test data set had to be acquired as a first step. For this 448 captchas were single-handedly solved and checked against the reCaptcha system. From those 448 solved puzzles, only 8.4% (38) were rejected, as the captchas were solved by a highly trained user. Based on the leftover 410 captchas, which will serve as the ground truth in this thesis and will be called the "captcha test set", we will now develop the model. Furthermore, we will try to find real world occurrences for these distortions.

[3] If not other noted, all reCaptcha images were generated by the reCaptcha API and might therefore be copyrighted by reCaptcha and/or Google.

2.3.1 Sloping

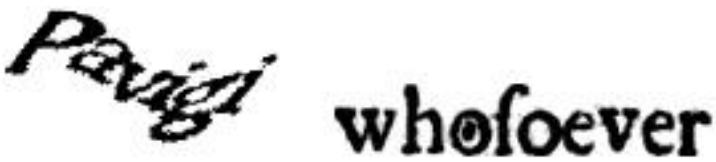


FIGURE 2.5: The baseline of the "Pavigi" challenge is offset by a curve, or rather the text is set on the curve.

The first most obvious distortion is the wavy form of the text. It look like it is written on a wave.

This is a common distortion in the real world. It can originate from wrinkled paper, curvature of paper, printing errors or heavy lens-distortion. It is also frequent for digitalized books where the inner part of the text is stretched to bond of the book. Another example would be creased or wrinkled paper or heavily wrinkled paper due to water damage. However, in the case of water damage also the ink would be affected.

This process can be modelled by offsetting the baseline of the text by a curve. While offsetting a given text on a curve one has two options to align the single characters on the curve. For one thing the characters can just be offset on the curve like a staircase (2.6a) or the individual characters can be rotated according to the local curvature (2.6b).

(A) Baseline offset by a staircase along the curvature.

(B) Characters set on the baseline rotated according to the local curvature.

FIGURE 2.6: Two possibilities for offsetting a text on a curve ("sloping").

While the term "Sloping" usually only refers to a straight, but angled baseline, we will still call this process "Sloping", since it refers to an anomaly of the baseline.

2.3.2 Slanting

FIGURE 2.7: Slanting of the word "hersoved". Notice the angled stems of the 'h' and 'd'.

The next anomaly is the slanting of the characters. While not being a real distortion it immensely hardens the process of segmenting the characters, since the individual characters are now overlapping.

The slanting occurs very often in the real world, the most well known would be *the itacilization* of text.

Remark 1. *Good font sets provide special italic-subfonts for real italic typesetting. They sometimes differ slightly in single characters e.g. a real italic version of & is & and not &.*

Yet, simple shearing is sometimes used (e.g. by Microsoft Word) and achieves roughly the same results.

We are able to model this by a simple shearing operation on the word itself.

Notwithstanding we must point out that slanting and sloping are not commutative. In order to correctly slant all characters they must all be set on the same, straight baseline. Thus, the sloping operation must be done after the shearing operation. I.e. we will first need to de-slope the image prior to de-slant the image.

2.3.3 Connection of Characters

The connection of the individual characters is the most difficult of the occurring distortions.

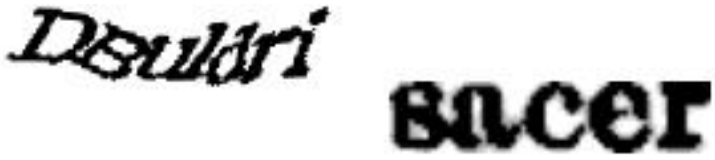


FIGURE 2.8: Interconnection of all characters. Notice that the characters are connected at different positions, most times by their shortest distance.

The digitalization of a text segment usually consists of a segmentation phase and a recognition phase [GWF12]. While in the early days recognizing the individual characters was the challenge, nowadays it can be considered the easier one (e.g. [YE07, YE09]). Thus, the real challenge is the segmentation of the characters.

With the connection of the characters this however is hardened immensely. We will later tackle this problem by combining the segmentation- with the recognition-phase.

Notice that the characters are not always single-connected, in some cases they are connected via two connections, thus forming intermediate holes which can be easily mistaken as new characters. (See Figure 2.9). In the real world this often appears on low quality prints. After a fresh print the

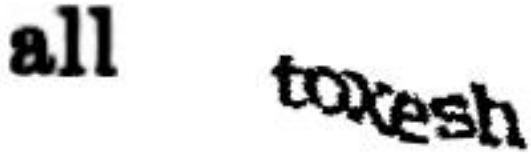


FIGURE 2.9: Double connected characters lead to intermediate holes. Thereby, one can also see a 'b' in the first character if one only focuses on the first part. The human brain automatically corrects this to a "to" once it sees the left over part after the ambiguous 'b' which can not be classified alone.

still wet ink could be smeared or it could bleed out on non-suitable paper. Also water damage and thus runny ink can cause such connections.

Notice that this connection is different from other used "connection-strategies" (e.g. Figure 2.10), where arbitrary strokes or random speckles are used. The direct connection of the characters more likely resembles some kind of handwriting or permanent, unremovable distortions on the paper itself. Arbitrary strokes resemble dust or hair on the paper that could be removed physically before the scan.

FIGURE 2.10: An old version of the Google reCaptcha system with arbitrary added strokes connecting the normally separated characters. Source: `http://blog.recaptcha.net/2008/12/funny-recaptchas.html`

Chapter 3

Word Processing

In this chapter we will process the complete captcha or subsegments of it in its entirety. We will first split the captcha into pro-bono and challenge and then further process the challenge part.

Our goal will be to process the challenge in a way that we will be able to separate the characters via straight-cuts, i.e. vertical snips with a scissor through the challenge. Yet, this is prevented by the overlap of the characters originating from the slanting/shearing of the characters.

In order to correct the overlap, we will need to de-slant the image. Yet, the direct de-slanting is prevented again by the wavy baseline.

Thus, we will first start with correcting the baseline to further correct the slanting of the image.

3.1 Separation of Pro-Bono and Challenge

As described in Section 2.2 the reCaptcha consists of two parts, the pro-bono part and the interesting challenge part. Thus, the first task will be to extract the challenge.

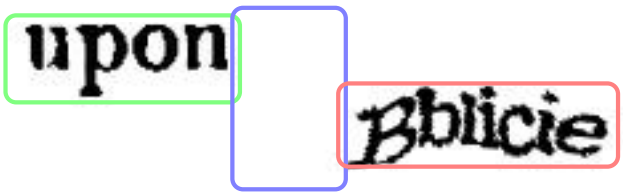


FIGURE 3.1: Separation of challenge part and pro-Bono part. The island of zeros (blue) clearly separates the challenge (red) and pro-bono (green) from each other. Furthermore, the challenge part is dimensionally higher than the pro-bono part, due to its curvature.[1]

Observations showed that both parts are always clearly separated by a free space, allowing us to separate them easily.

We will do so by looking at the projection of the image onto the x-axis via the sum of pixels per column. We will call this the horizontal projection of the captcha. In this projection we will search for the longest period of zeros in the middle, giving us the point where we need to segment.

Further observations showed that almost always the challenge part was taller (dimension wise) than the pro-bono part. Thus, the first part of the algorithm for extracting the challenge part is as follows in pseudo code:

[1] Image processing is performed on a logical-image. Meaning the text is logical 1 and the background logical 0, which results in a "white-on-black"-image. For printing-purposes all images are therefore displayed negative as "black-on-white".

LISTING 3.1: Separation algorithm – Extracting the challenge

```
function extractChallenge(image){
  //extracts the challenge out of the captcha

  //remove white spaces at the border
  //shrink to smallest bounding rectangle
  image ← imcrop_total(image);

  //project the image
  projection ← sum(image, 'per Column');

  [islandIndex, islandLength] ← findIslandOfZeros(projection);
  longestIsland ← argmax(islandLength);
  cuttingLocation ← islandIndex(longestIsland);

  [left, right] ← cutImageAt(cuttingLocation);

  if(left.size.h > right.size.h)
    return left;
  else
    return right;
}
```

3.2 De-Sloping

In order to correct the curved baseline, we will follow a very simple approach similar to that used in [GWF12, Weg10].

The baseline is the line on which almost all characters are aligned with their lowest point. This holds true except for the characters 'p', 'q', 'g' and 'y'. We therefore just calculate the lowest set pixel per row (Figure 3.2left) and then heavily smooth this line to prevent rapid jumps between or inside the characters (Figure 3.2middle). This also removes the rapid jumps to below the baseline formed by 'p', 'q', 'g' and 'y'. For smoothing we use a robust version of *locally weighted scatterplot smoothing* (RLOWESS), which adds a higher weight to outliers.

As one can see in Figure 3.2(right) even this simple approach yields great results. Still, it might sometimes run into problems.

FIGURE 3.2: The simple desloping process. From left to right: The lowest pixel per column is calculated, the baseline is heavily smoothed, every column is shifted downwards by the baseline. Note that even this simple approach delivers great results.

The proposed scheme works great as long as it has some fix points at the start and at the end—further called the "edges" of the word.

Anyhow, as soon as the fix points are missing, the calculated baseline will drift away at the edges (see Figure 3.3).

FIGURE 3.3: The simple desloping process sometimes runs into problems as one can see from this example. The baseline drifts away at the edge, since it has no fixpoints to stay at. This heavily destroys the last 'd'.

To prevent this we will hold down at the edges to the last plateau. All characters form at least a small plateau at their correct baseline. With this technique (Figure 3.4) we can correctly identify the baseline in this setting.

FIGURE 3.4: The more sophisticated approach by keeping down to the strongest plateau at the edges. Note that it no longer destroys the last 'd'.

Considering slanting of the characters of serif fonts sadly disables the "use the first plateau" approach. A slanted T or F would form a clear plateau at the edges. Also a serif 'H' (like in Figure 3.5), forms a clear plateau already right at the beginning. The problem becomes even worse once we

FIGURE 3.5: Selecting the plateau to stick to is not trivial. Notice that the upper serif of the 'H' already forms two (rather small) plateaus right at the beginning. Therefore, the deepest of the longest plateaus is selected.

consider noise, where arbitrary noise-speckles might form random (small) plateaus right in the middle of a character.

We therefore will take the deepest plateau of the three longest plateaus at the edges. We define the edges as a fourth of the image from the left and right.

This approach might sometimes select a too deep-plateau, such that the shift of the first or last characters is not corrected, as can be seen in Figure 3.6. While this will affect our de-slanting algorithm it is not as bad as a wrongly identified baseline. One can imagine that the destroyed character from Figure 3.3 is worse than the slightly uncorrected baselines in Figure 3.6.

FIGURE 3.6: Selecting the deepest of the longest plateaus at the edges sometimes selects a too deep one, leading to a slightly uncorrected baseline at the edges ('ol' and 'ca'). Still, this is not as bad as the previously seen destroyed 'd' in Figure 3.3.

As the characters of a not completely corrected baseline still remain intact and readable we will accept this issue over the risk of completely destroying single characters.

Listing 3.2: Desloping Algorithm in Pseudo-Code

```
function deslopeImage(image){
  [h,w] ← size(image);
  baseline ← [0]^w;

  foreach(column x in image)
    baseline(x) ← findLowestSetPixelInColumn(x);

  //Perform slight smoothing
  smooth(baseline, 3);

  //Stick to Plateaus
  leftPart  ← [start → w·0.25];
  rightPart ← [w·0.75 → end];

  //find longest
  leftPlateaus  ← findLongestPlateaus(image(leftPart), 3 plateaus);
  rightPlateaus ← findLongestPlateaus(image(rightPart), 3);

  //find deepest
  stickToLeft  ← min(leftPlateaus, 'y');
  stickToRight ← min(rightPlateaus, 'y');

  //Stick to this plateau
  baseline(start → stickToLeft.Index)   ←  stickToLeft.Value;
  baseline(stickToRight.Index → end)    ←  stickToRight.Value;

  //Perform Heavy Smoothing
  smooth(baseline, 50);

  //Ensure still sticking to plateaus
  baseline(upto stickToLeft.Index)     ←  stickToLeft.Value;
  baseline(from stickToRight.Index)    ←  stickToRight.Value;


  //Deslope image
  deslopedImage ← image;
  foreach(column x in image)  //Loop through all coordinates
    foreach(row y in column)  //Shift all (set) pixels by baseline in column
      deslopedImage(y + h−baseline(x),x) ← image(y,x);

  return deslopedImage;
}
```

3.3 De-Slanting

As we have straightened the baseline, we are now able to correct the slanting of the characters. This is also a frequent problem in handwriting recognition, where some writers tend to write slanted to the left and some to the right. We will refer to an algorithm proposed by Kavallieratou et al. in [KFK00, KFK01], with slight modifications.

The basic idea is again very simple. Many characters contain clear vertical lines ("stems"), e.g. [T t m n b B], etc. Once these get slanted like [$T\,t\,m\,n\,b\,B$], the lines still remain, but are not vertical anymore.

When considering the x-projection as the sum of pixels per row again, this leads to a very neat effect. With straight, vertical lines the sum of pixels per row is maximized, they form peaks. But once they get slanted they distribute over the neighbouring characters, leading to a significantly flatter x-projection. For visualization consider Figure 3.7, clearly showing the dominant peaks in the deslanted version, which vanish in the slanted version.

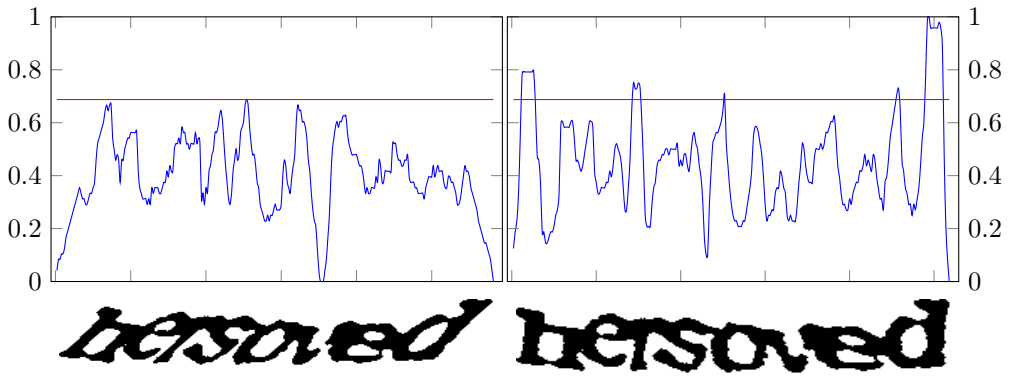


FIGURE 3.7: The slanted (left) and deslanted (right) versions of the "hersoved"-captcha. The projections are normalized to the highest peak of the deslanted version. Notice the well defined and high peaks present in the deslanted version. Notice further the double maxima formed in some peaks which need to be ignored.

In order to show that this also holds for different shearing angles, Figure 3.8 shows a simulation of different shearing angles on a normal text segment.

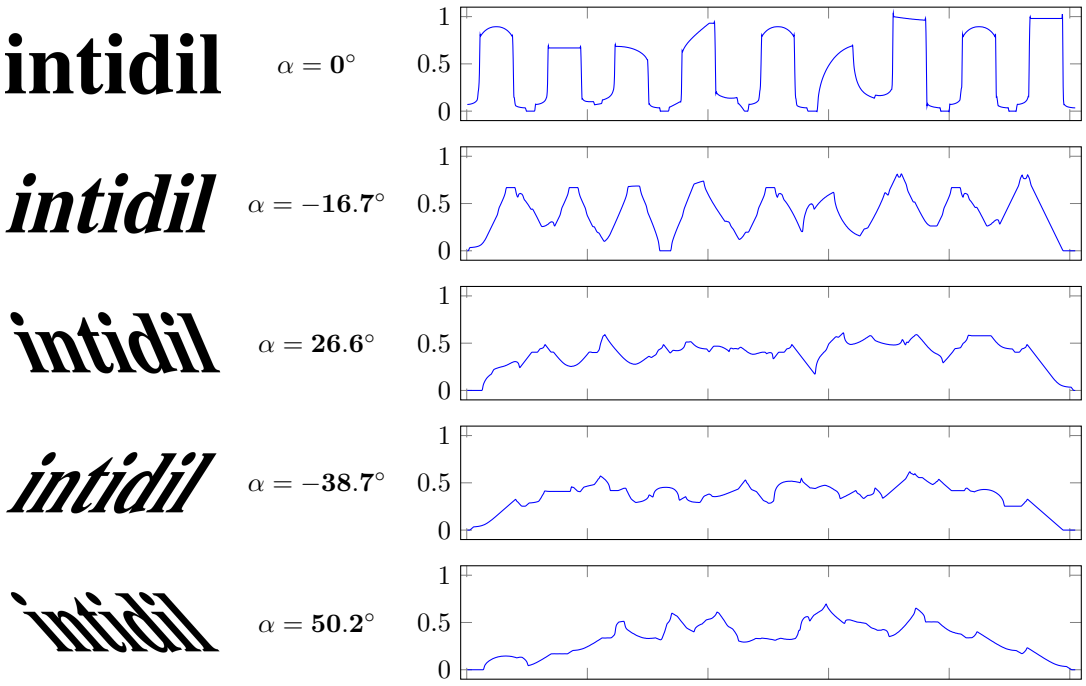


FIGURE 3.8: Simulation of different shearing angles affecting the horizontal projection. Notice the extremely distinct peaks of high amplitude in the unsheared case that fade and blur rapidly opon shearing.

We will therefore search for the shearing-angle that will lead to the "peakiest" x-projection. To find this "peakiness" Kavallieratou et al. [KFK00, KFK01] utilized the so called Wigner-Ville-Distribution. We however will follow a simpler approach. As a measurement for peakiness we will take the median of the five highest peaks. Deciding the number of peaks to consider depends on the average word length and the frequency of stem-characters in this language.

While this sounds very easy, noise on the other hand might lead to some complications. During a peak, random noise might interrupt the peak and lead to a small fluctuation downwards. This then will break one peak into multiple maxima, where we would normally just expect one, formed

by one vertical line (see Figure 3.7). In order to not count these twice, a minimal distance between two maxima is required. While scanning all maxima from left to right, a maximum blocks following maxima for a certain blocking range.

The blocking range of course depends on the average width of a stem. Empirical studies have shown that a range of ± 10 pixels shows satisfying results.

Listing 3.3: Deslanting Algorithm in Pseudo-Code

```
function deslantImage(image){

  projection ← sum(image, 'perColumn');

  //BruteForce all Shearing Angles
  for(shearingAngle from −1.8 to 1.8, Step 0.05){
    tmpImage ← shear(image, shearingAngle);
    tmpProjection ← sum(tmpImage, 'perColumn');

    tmpMax ← getHighestPeaksMean(tmpProjection, block for 10 pixels, get 5 peaks↩
        );

    //Remember the Maximum
    if(tmpMax > cMax){
      maxMean  ← tmpMax;
      maxAngle ← shearingAngle;
    }
  }

  //Only shear if reasonably improved
  if(maxMean < 1.05·getHighestPeaksMean(projection, 10, 5))
    return image;

  //return sheared Image
  return shear(image, maxAngle);
}

function getHighestPeaksMean(projection, blocking, n){
  //collects n maxima from projection.
  //calculates the mean.
  //blocks around every maxima for >blocking< pixels.

  maxima ← extrema(projection, 'maxima');

  foreach(maximum in maxima, order by x−ascending){
    if(last maximum nearer than blocking)
      remove maximum from maxima;
  }

  sort(maxima, by y descending);

  //return median of n highest maxima
  return median(maxima(1:n));

}
```

3.4 Evaluation

With the desloped and deslanted image it is now possible to segment the word into single characters via straight cuts.

In order to evaluate the performance of the proposed desloping- and deslanting-algorithm, the captcha test set of 410 captchas was processed by the algorithm and presented to a human. With the now straightened words, the human placed the correct cutting positions.

This evaluation served two purposes. On the one hand it tested the proposed algorithms, on the other hand it generated a great test set of heavily distorted single characters. This test set will come in very useful when designing and evaluating our classification engine for single characters.

All told the proposed algorithm delivered very satisfying results. Some selected examples are presented in Figure 3.9.

Out of 410 captchas, only eleven captchas were separated wrong into pro-bono and challenge part and an astonishing amount of 87.3% (358 captchas) were separable via straight cuts. Leaving only 41 captchas left that were not separable, of which about 50% were due to a wrong slant correction and only in six cases the desloping algorithm failed completely. It should be pointed out that some words are not desloped completely; some curvature might still remain. Certainly this is no problem, as long as the characters are separable via straight cuts. These numbers are also represented in Table 3.1.

TABLE 3.1: Manual evaluation results of the desloping- and deslanting-algorithm.

#	**Result/Reason**	%
410	Captchas	100.0%
358	Separable	87.3%
11	Wrong challenge extracted	2.7%
41	Not Cuttable	10.0%
⋮ 6	Desloping failed	⋮ 14.6%
⋮ 21	Deslanting failed	⋮ 51.2%
⋮ 14	Other	⋮ 34.1%

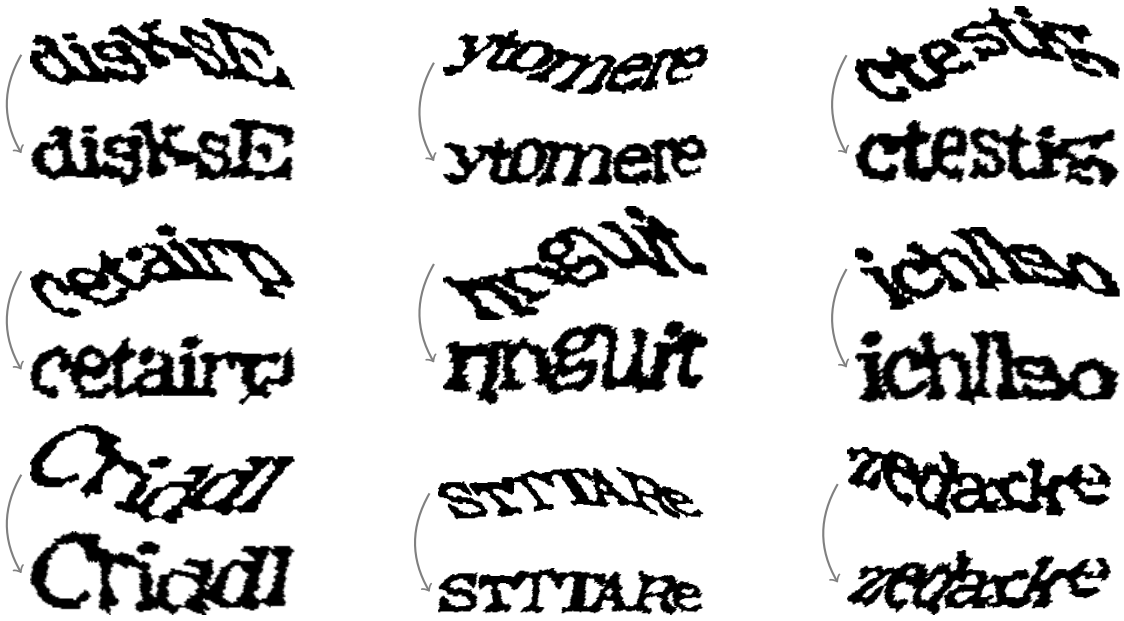


FIGURE 3.9: Results of the word-processing algorithms. Notice the last result, where the deslanting failed and slanted the word too much.

Chapter 4

(Over)Segmentation

While the previous chapter considered the captchas as complete words, the next chapter will focus solely on single characters and their classification. This chapter will build the bridge between both of them by supplying a segmentation algorithm that will cut the complete word into single characters.

We will not force the segmentation algorithm to solely cut at the desired, correct positions, i.e. also cuts at unnecessary positions are permitted—it will oversegment the word.

Later on, the correct segmentation positions will be determined via a backtracking algorithm based on the classification-certainty of the individual segments.

4.1 Separated Segments

Prior to heavily oversegmenting every individual word, we shall rescue already separated characters or word segments. As an exemplar one can consider the "Bblicie"-captcha (Figure 4.1) from the previous sections, where the 'B', 'b', 'lici' and 'e' are already separated by not having a connection with the other characters or segments. As one can further see, a straight cut between 'c' and 'h' of the word "DGraSch" (Figure 4.1) would however chop off some small parts of the h. We will therefore stretch already separated segments far apart, disallowing a further merging of these segments.

However, as one can see from Figure 4.1(right), not every separated segment is an isolated character or segment. In this case the upper part of the 'G' broke off and forms another isolated segment. This can often happen on thin and fragile lines of characters. Referring to Figure 4.1(left) one may notice that those segments are not necessarily broken characters but can also be formed by i-dots. We therefore need to detect that some segments might belong to another group of the image and are not a separation of two word segments.

To do so, at first all isolated regions not connected by an eight-neighbourhood are labelled. Further, the bounding rectangles of all regions are checked for overlapping along the x-axis (horizontally). Once a segment is overlapping horizontally more than 50% of its width with another region it will be relabelled into this region. Afterwards the individual regions are stretched apart, i.e. processed as separate words.

FIGURE 4.1: Parts of the challenges are already separated, i.e. not connected.

FIGURE 4.2: After initial labelling of connected regions, the algorithms detects whether single segments belong to another (bigger) group. Following it stretches the segments far apart, such that they are not merged in further processes.

LISTING 4.1: Stretching Algorithm

```
function stretchImage(image){
  //finds belonging components.
  //stretches separate regions far apart.

  removeObjects(image, 'smaller than 36 Pixels');

  labelConnectedComponents(image);

  //Relabel related components
  foreach(component a in image){
    foreach(component b in image){
      if(a.x-Range lies 50% in b.x-Range)
        a ← b; //relabel
    }
  }

  //Stretch far apart (or process separate)
  foreach(component c in image){
    stretchedImage ← stretchedImage.append([c space]);
  }

  return stretchedImage;
}
```

This would also be a place where an algorithm for reconstructing broken characters could be implemented. Whether or not and how this could be implemented will be discussed now.

4.1.1 Broken, Over- and Under-Connected Characters

In the previous section we have seen that some characters might break apart, leaving shards at the previous places (e.g. Figure 4.3b). These shards give a hint that a character might be broken and needs fixing. Yet, implementing such a fixing algorithm is not trivial.

Moreover, during the manual cutting of all captchas more observations have been made. Besides broken characters with left over shards, some characters were also broken without leaving any shards at all. Figure 4.3a shows some examples of these under connected characters. A possible real world explanation for this could be scrapped of ink from the paper or a low resolution scanning. On top of that, some characters tend also to be overconnected with themselves, a problem that might originate from the same effects that also connect intermediate characters. However, these connections (e.g. Figure 4.3c) form completely different characters and are (without context) not recognizable anymore.

(A) Underconnected (B) Broken with shards (C) Overconnected

FIGURE 4.3: Different types of degraded characters discussed in this section.

FIGURE 4.4: The 'B' blocks the caliper from measuring the lean connection between 'e' and 'B'.

All in all it becomes clear that broken characters are not easy to reconstruct. An analogy would be the purchase of a heavily used vase. With the presence of some shards one at least knows that the vase is broken and needs to be fixed, but one does not immediately know where exactly the shards should go. However, with the absence of shards one has no indication whether the vase is broken or not; even an unusual hole in it might be the choice of the artist himself. The same is applicable to the interconnection, where an unusual connection might originate from some manufacturing defect or might again be the choice of the designer—a different character.

It becomes clear that only the broken characters with left over shards might be reconstructible. Since we already identified to which part the segment belongs to, it could be connected via its shortest distance to it. Still, one would need to distinguish between i-dots and shards that need to be connected. Moreover, the shortest distance is not always the best choice, as the shard of the 'G' in 4.1 (right) would then be connected to the 'r'. [Dro03] proposes an algorithm which might solve this problem, by connecting the shard to multiple candidates and then further checking the quality of the classification. Yet, this would introduce another layer of backtracking in our algorithm, as we already will use this for the identification of correct segmentation points already.

After these remarks we will refrain from considering broken characters in this thesis as it would be beyond the scope of it. Furthermore, with only 91 instances out of 2400 the amount of broken characters with left over shards is rather small. Also the amount of 225 instances of characters that are under- or over-connected is significant, but still rather small.

Nevertheless, this remains a very interesting research topic.

A note on morphological operations In image processing a very powerful tool are morphological operations. Basic operations include dilation and erosion, as well as closing and opening. With a closing operation small holes or gaps can be filled, in contrary a opening operation breaks loosely connected structures. Basically this could be used for reconstructing degraded characters. Wrongly connected characters could benefit from an opening and underconnected characters from closing operation. Anyhow a closing operation would also connect nearly connected structures, like the jolt of an 's'. The same effect occurs for an opening, which would break correct, thin connections of a character.

All in all morphological operations would "heal" some degraded characters, but at the same time destroy various fragile proper characters. We will therefore not utilize morphological operations in the classification or segmentation engine.

4.2 Oversegmentation

Since the word now consists of only inter-connected characters, we will need to find a way to identify possible cutting points.

Some papers (e.g. [LSA94]) utilize a so called "Caliper-Distance" for this. Named after the tool (Caliper, german "Messschieber"), it measures the distance of lowest and highest point per row and further cuts at the closest points.

While this works great on straight, but connected characters, it will fail on such heavily distorted segments we are considering in this thesis. As one can see from 4.4 the caliper distance works at the separation of 'a' and 'l', but fails on the separation of 'e' and 'B', since the serif of the big 'B' blocks the highest point from meeting the lowest point at this row.

We will therefore utilize again the projection of the image onto the x-axis by the count of pixels per column.

As one can imagine possible cutting positions always contain a very small number of pixels per column and therefore form minima in the projection. Therefore, all minima will be considered as candidates for cutting points.

To further increase the precision we will prevent cuts through holes in characters. This will enable that characters like 'b' are not intermediately cut, although they form a minimum after their stem. To prevent this, all holes are filled prior to calculating the projection. Due to the fact that adjacent characters with two touching parts might also build holes (remember the "toxesh"-captcha from Figure 2.9), we will further formulate the requirement that the holes to be filled must be at least some kind of circular. This is checked via calculating the extent, the ratio of area to bounding rectangle. We set the requirement that the extent must be larger than 50% and further that the width must be larger than half of the height.

Considering the previous projections from the "deslanting"-section and Figure 4.5, one can see that even the simple approach taking all minima as cutting hypotheses already provides good results.

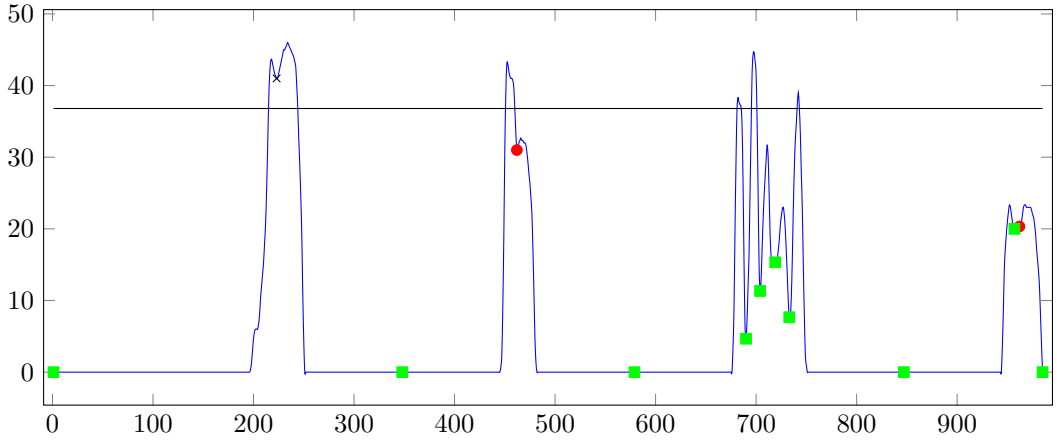


FIGURE 4.5: Horizontal projection of the 'Bblicie'-Captcha and the detected cutting-positions hypotheses. Minima deleted due to not lying inside a valley are marked red. Too adjacent minima or minima lying above the threshold are marked black. The final hypotheses are marked in green.

For all that, small noise fluctuations lead to small and short minima that need to, or rather, can be disregarded. A first idea was to consider the second derivative at the minima for changes of the slope above 1 in an adjacent region. While this approach showed acceptable results, sometimes small minima were not deleted and, even worse, some clear and correct minima were deleted. With this approach only very steep and rapid minima are selected and slowly descending minima were disregarded.

Therefore, a second requirement for being a minimum is added. Each minimum must lie in a valley, surrounded by steep walls. Therefore, an adjacent area of ± 8 pixels is scanned and checked for the maximum value to the left and right. The difference of the lowest maximum to the possible candidate must be at least greater than 2 pixels. A very small, but very effective threshold, efficiently eliminating small noise fluctuations along a descend or ascend.

Further, the cutting positions are not allowed to be too close together. Therefore, a bottom-up scanning process will eliminate close minima, keeping only the deepest of neighbouring minima.

Finally, this leads to a hypotheses of possible cutting points as shown in Figure 4.6. Adjacent segments will later be merged and checked against the classification engine, which we will now develop.

FIGURE 4.6: Potential cutting hypotheses of the 'Bblicie'-Captcha found by the algorithm. Notice the cuts at all necessary positions, but also at unnecessary, which will later be merged.

LISTING 4.2: Over-Segmentation Algorithm

```
function segmentImage(image){

  projection ← sum(image, 'per Column');

  //smooth very gently
  projection ← smooth(projection, 3);

  //find minima
  minima ← extrema(projection, 'minima');

  //require minima to lie in valleys
  foreach(minimum in minima){
    leftMax  ← findNearestMaximum(upto 8 pixels, 'to left');
    rightMax ← findNearestMaximum(upto 8 pixels, 'to right');
    lowestMax ← min(leftMax, rightMax);

    //if no valley of 2 pixels is present
    //and it is not an absolute minimum (very few pixels in column)
    if(lowestMax − minimum < 2 ∧ minimum > 10)
      remove minimum from minima;
  }

  //require that minima don't lie too close to each other
  foreach(minimum in minima, order by y−descending){
    //delete adjacent minima (keeping the highest, since it's ordered descending)
    removeAdjacentMinima(around ±6 pixels);

    //disallow too high minima
    if(minimum > 80%· max(projection))
      remove minimum;
  }

  return minima;
}
```

Chapter 5

Character Classification

In this chapter we will deal with the classification—or recognition—of single separated characters.

For gathering the required test and learning data, we will use the previously separated characters from Section 4.2. From nearly 400 captchas, about 2400 characters were extracted and will be used here.

As said before the final recognition engine will recursively backtrack the oversegmented parts into the correct segments. This means our classification engine will need to serve two purposes. On the one hand it must be able to correctly classify even heavily distorted characters, on the other hand it must deliver a measure of confidence in this classification, such that non-character segments can be discarded.

While some current OCR systems are able to cope with the first requirement and can correctly classify distorted characters, we were unable to find any system that provides a fine-granular and reliable confidence measure.[1]

This chapter deals with the main part of this thesis, since without a very robust and precise recognition engine also the complete segmentation engine would be of no avail. Furthermore, we will see that we will develop a classification system for single characters that will beat current and most sophisticated recognition engines significantly.

5.1 First Ideas

Some earlier captcha systems used very easy, fixed character sets consisting of a fixed font-height and weight as well as being very crisp. Some of them were even detectable by the count of pixels alone [YE09, GWF12, YE07].

Encouraged by such an easy approach a first idea was to just take a simple **pixel-by-pixel difference** measure against a predefined set of characters; like the letters from the font-family "Times New Roman". The letter with the least pixel-difference would win.

While being very easy to implement and at first glance showed acceptable result, it soon got clear that this was a too naïve approach.

The first problem was how to scale the test and reference letter against each other. Scaling them to a common width and height would destroy valuable information about the shape—an 'i' would be blown up to an 'o' and almost anything would be scaled down to a small line when compared against an 'i'. Scaling all to a fixed width and height destroyed valuable information on proportions.

[1] Tesseract provided a confidence measure up to version 3.01, but it was not documented well and only provided a 3 bit confidence measure. As of version 3.02 it is only available by patching the source code and recompiling. [Cod]

Moreover, varying font-weight (or pen stroke) completely destroyed the measurement. For example an 's' scaled up to a 'B' fits very well in it, with just some errors on the connection of the upper and lower circle. On the other hand a slightly more thicker 'B', or a 'B' with some left over artefact (and thus not perfectly aligned on a reference 'B') resulted in a much higher error than an accidental "fitting" s.

All these problems lead to the conclusion that this approach would be of no avail.

Coping with the problem of true negatives (like an 'S' against 'B') and false negatives (like thick 'B' against thin 'B'), the idea was to implement a **weighted error** based on the distance to the next pixel that would fit. While being hard to implement it showed better results. Nevertheless, the problem of alignment still existed and it was computationally too expensive.

Discarding all the previous ideas, the idea for using a **neuronal network** arose. With the amount of activation per output neuron it would hopefully deliver a confidence measure. However, while being hard to configure and determining the number of layers, hidden layers and input neurons it also needs a big set of inputs.

Since our goal was to not tailor or algorithm too hard to a given system, we will need to discard this idea too. Also slight changes in the character set would always mean relearning the complete neuronal network. Furthermore, a neuronal network is always a way of non-deterministic and leads to very unexpected results when confronted with completely different data (like non-characters from wrongly merged segments).

We therefore focused on the nature of the characters themselves. The human is able to read almost any arbitrary font-type irrespective of its weight, stroke-width or scaling. We will therefore try in the following to resemble this process by a heavy feature extraction on the characters.

5.2 Feature Extraction

Humans are able to read almost any font face, irrespective of their shape, curvature, decoration (serif, sans-serif) or scaling. Even handwritten characters are recognizable, even without having any font-weight at all, they are just a skeleton of the character.

We will therefore now try to extract the features of letters that enable this process.

5.2.1 Character Shape

The most important feature of a character is its shape, and outline or boundary. Basically this shape is always the same, independent of the used font-face and -weight or even handwriting. Between different font-types only small decorations change—they can be serif or sans-serif, some use intermediate lighter-strokes, some have a constant width—but the outer shape always stays the same.

Thus, for comparing the shape of a character we need to generalize it. We shall not care about small interrupts due to noise or serifs. We will want to smooth it deriving the heart or essential structure that makes this outline unique.

In order to achieve this we will use so called Fourier descriptors, they can be thought of as a Fourier series decomposition of two-dimensional outlines. Basically these transform the outline into an ordered sequence of n complex numbers, which are then easily comparable to other (reference) sequences. Just as Fourier series they have the nice property to hold the basic structure in the low-frequency parts and the details in the upper high-frequency coefficients.

We therefore can heavily smooth (or generalize) the outline by discarding high frequency components and reconstructing the outline from the remaining low frequencies. 5.1 shows a reconstruction from the Fourier descriptors for the letter 'F' at different approximations.

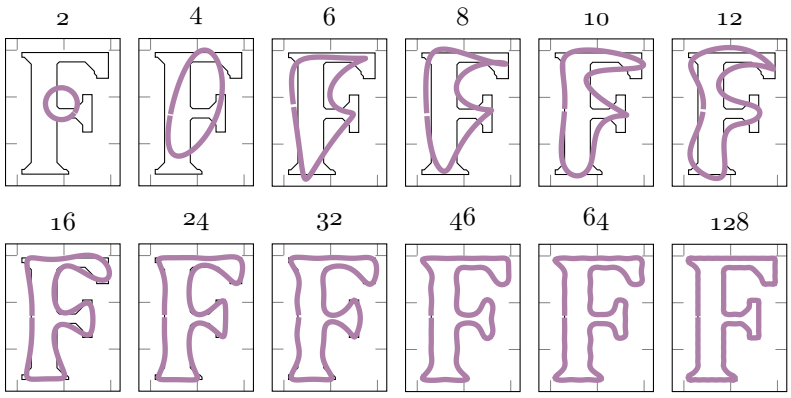


FIGURE 5.1: Reconstructed 'F' by different numbers of Fourier descriptors.

To get a first impression of the generalization ability, Figure 5.2 shows the comparison of a heavily distorted 'F' from a captcha and the corresponding 'F' rendered directly from Times New Roman. Note that in this image the rotation, scaling and translation is not yet eliminated, but only focuses on the generalization of the outline.

FIGURE 5.2: Comparison of distorted and reference capital F. Notice how the reduced Fourier descriptors (16) heavily generalize the shape of the letters.

The exact calculation of Fourier descriptors and the mathematical derivations of their properties can be found in the appendix A.1 on page 53. For now we will take the computation and their properties as granted.

Invariants of Fourier Descriptors

Fourier descriptors are especially interesting for us, since they provide a heavy degree of invariance against rotation, scaling, flipping, translation and the starting point of the boundary. The derivations are obvious and given in the appendix.

The translation and scaling invariance is achieved by ignoring the DC component and normalizing all descriptors by the magnitude of the first component.

The remaining invariances are achieved by just considering the absolute value, since only the phase is affected by rotation, flipping and starting point changes.

In order to get a feeling for the power of Fourier descriptors and their high invariance, we will consider several rotated, mirrored and scaled 'd's. For comparison a slightly similar, but different 'o' and two completely different 'T' and 's' are given. Notice that the 'o' almost has the same outline, except a short trip around the stem of the 'd'. The first 16 Fourier descriptors for these letters are given in Table 5.1.

As one can see from Table 5.1 the absolute values are almost constant, even on heavy scaling and rotation. Also, while being similar in shape, the 'o' shows very different absolute values as well as the different 'T' and 's'.

TABLE 5.1: The first 16 Fourier descriptors for several distorted 'd'. For comparison different characters 'o', 'T' and 's' are given.

d	180°	↕	↔	+15°	-30°	scaled	o	T	s
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
0.14	0.14	0.14	0.14	0.15	0.15	0.12	0.01	0.03	0.05
0.11	0.12	0.11	0.12	0.11	0.09	0.12	0.02	0.22	0.50
0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.00	0.02	0.03
0.03	0.03	0.03	0.03	0.03	0.04	0.03	0.02	0.01	0.04
0.03	0.03	0.03	0.03	0.03	0.01	0.02	0.00	0.08	0.01
0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.04
0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.00	0.02	0.01
0.01	0.01	0.02	0.01	0.01	0.02	0.03	0.00	0.08	0.06
0.03	0.04	0.03	0.04	0.03	0.03	0.03	0.00	0.01	0.00
0.01	0.01	0.01	0.01	0.00	0.03	0.02	0.00	0.02	0.14
0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.00	0.09	0.00
0.02	0.01	0.02	0.01	0.03	0.05	0.03	0.02	0.12	0.30
0.16	0.16	0.16	0.16	0.15	0.15	0.17	0.00	0.42	0.03
0.23	0.24	0.23	0.24	0.23	0.23	0.13	0.03	0.26	0.55

Like stated before the rotation only manifests itself in the phase angle. Empirical studies have shown that the rotation is approximately linearly correlated with the phase of the first coefficient. Mirroring along x and y leads to a complex conjugation in some of the phase angles.

As one can see from the mathematical derivations in the appendix, a rotation should affect all phases at the same amount. Also all phases should be complex conjugated on mirroring along x. A mirroring along y should lead to a complex conjugation of the phase in addition to a phase shift of π. Finally, a rotation of 180° should lead to a constant phase change of π.

Sadly though this does not hold true in practice, because, in addition to the rotation and mirroring, the starting point of the boundary is also changing. The changing of the starting point leads to a linear change in the phases. The starting point always changes, because it is in our implementation fixed to always start at the left centre. That's why the theoretical phase changes are not directly evident in practice. Luckily as noted above they are still evident in some of the phases.

If one would implement a constant starting point (like always starting at the end of a stem) it should be possible to utilize the phase for a very precise orientation measurement. Though without knowing anything about the character yet, it is quite impossible to find a constant feature for starting point selection. (One would need a feature for starting point selection that selects the same point on all characters, without knowing what character it is. For example it would need to select the same point among [b d p q], [c u n], [s S N] or [t f i l] and further also search in the same direction.)

With that being said the heavy invariance of the Fourier descriptors shows up to be a problem in character recognition. *Characters are per se not invariant.* A 'p' mirrored along the x-axis looks like a 'b', y-mirrored like 'q' and xy-mirrored like 'd'. For comparison Table 5.2 shows these four letters [b d p q] side by side.

This also happens in some other cases. For example a jolted 's' with a tight curvature of the upper curve, has approximately the same shape as a 90° rotated 'n' or always has high similarities to a 'Z' or 'N'. Furthermore, the characters 'u', 'n' and 'c' are almost identically in shape, but only rotated by some angle.

We therefore need to also consider the phases on variant characters in which the information about mirroring and rotation (i.e. orientation) is partially enclosed.

TABLE 5.2: Comparison of the Fourier descriptors for the letters 'b','d','p' and 'q'. Notice they are approximately constant and do not differ by magnitudes as they should normally (e.g. from 'T').

sml q	sml b	sml d	sml p	big T
0.00	0.00	0.00	0.00	0.00
1.00	1.00	1.00	1.00	1.00
0.10	0.13	0.14	0.14	0.03
0.09	0.12	0.11	0.11	0.22
0.05	0.08	0.07	0.06	0.02
0.05	0.02	0.03	0.05	0.01
0.01	0.03	0.03	0.01	0.08
0.00	0.00	0.00	0.01	0.00
0.01	0.01	0.01	0.01	0.02
0.01	0.02	0.01	0.02	0.08
0.00	0.04	0.03	0.01	0.01
0.03	0.02	0.01	0.03	0.02
0.01	0.05	0.05	0.02	0.09
0.07	0.01	0.02	0.06	0.12
0.13	0.16	0.16	0.14	0.42
0.12	0.25	0.23	0.14	0.26

We will try to prevent a too high rotation invariance by checking the phase of the first coefficient. Mirroring often leads to a complex conjugation in the phases of coefficient two and three. While the phase check on the first coefficient can be done by a simple threshold of e.g. $45°$, detecting complex conjugation can also be quite tricky. A complex conjugation of $a\angle 5°$ leads to $a\angle -5°$, a phase change of only $10°$, which is the same result as simple rotation of the character by $10°$. So even if one would find the holy grail of starting point selection, one would still struggle with detecting the complex conjugation originating from mirroring.

In order to still utilize the phases, while keeping some rotation invariance, we will discard phase differences in the second and third coefficient of more than $100°(\approx 2\text{rad})$. All other phases are discarded as it was derived in the appendix.

Remark 2. *This phase check can be disabled on per se invariant characters like [o O D i I l j A]. An 'A' could be confused with a 'V', but includes a hole.*

Remark 3. *While implementing keep in mind the jump from π to $-\pi$ in the complex phase angle.*

Still, as mentioned above, the shifting of the starting points and small complex conjugations lead to a significant amount of wrong classifications based on rotation variance. Empirical studies showed up that especially a significant amount of [b d p P q], [c C u U v V n] and [s S N] were confused. Most annoyingly many 's' were mistaken for an 'N'. We will therefore extract further features dealing as necessary conditions, with the Fourier descriptors dealing as the sufficient condition. I.e. without the necessary condition being fulfilled for the candidate, we do not even have to check whether the Fourier descriptors match the general shape. This can also be thought of as a thinning of the possible neighbours space.

5.2.2 Feature Removal and Detection of i-Dots

Prior to any calculation or detection of necessary and sufficient conditions the segments must be cleaned. It is very common that adjacent characters reach into the field of another character. This can be due to wrong or inaccurate segmentation or due to typographic exceptions like kerning or ligatures.

It is therefore possible that the classification engine is confronted with an image consisting of multiple parts. As discussed earlier, we can assume that multiple parts are not part of one character, as we will ignore broken characters. Indeed one exception exists, naturally 'i' and 'j' consist of multiple parts—their dots. In typographic terms this dot is called a "tittle", but we will simply refer to it as an "i-dot".

We will therefore only analyse the biggest connected object of an presented segment. During the search for the biggest object we will also detect possible i-dots, as a segment whose centroid lies completely above the bounding rectangle of the biggest connected object.

Of course the potential i-dot can also be a left over artefact of a neighbouring character and therefore be a false-positive i-dot. It will only deal as a slight indicator for the character to be an 'i' or 'j'.

5.2.3 Necessary Condition – Height and Minimal Area

Since the text segment is heavily oversegmented, it is possible that a partial segment, which is way too small to be a character, is presented to the algorithm. Due to the fact that the Fourier descriptors posses the before mentioned invariances, these would still be recognized as *some* character.

FIGURE 5.3: Some heavily oversegmented parts need to be ignored (coloured), since they are too small to form a character and the Fourier descriptors would still match *some* character. Notice how the green part forms something similar to an 'c'. This image further shows that the classification engine might be confronted with multiple objects in one segment; the biggest object is then chosen.

To ignore these false segments, at first all objects with an area of below 36 pixels (that is for e.g. a very small 6x6 Square) are deleted. Furthermore, the remaining largest object from the previous feature removal step must be at least taller (dimensional) than a given threshold.

Empirical studies have shown that a minimum character height of 28 pixels achieved satisfying results, accepting all valid characters and ignoring false partial segments.

A segment, which is too small by the above criterion is ignored during the backtracking process. It is given a weight of ∞, such that the algorithm is given no choice in choosing this segment alone.

Anyhow one small exception remains. Given a serif font it is natural from our oversegmentation algorithm that the serifs at the beginning of a word can be cut off (see Figure 5.4). Since these are very small objects, they are detected as an invalid segment by the above mentioned process. Of course there is no problem in ignoring these beginning serifs. The above mentioned "disallow too small segments" is therefore deactivated on segments at the beginning of the word or at the beginning of an stretched afar segment.

FIGURE 5.4: The oversegmentation chops off very small serifs (red) at the beginning of words. Normally these are detected as illegal cutting positions, certainly in the case of the beginning of a word, these small segments can be ignored safely.

5.2.4 Necessary Condition – Open Counters

In order to distinguish between letters like 'c', 'u' and 'n' we will detect the orientation of their open counter. In typographic terms an open counter (or aperture, German "Punze") is e.g. the arch found in 'c', 'n' or 'u'.

In order to detect the orientation of a possible arch, probes are sent from all directions into the candidate. The side in which the probe can dunk in the most will deal as the orientation of the character. E.g. to classify as a 'c', the probe must dunk in coming from the right.

To detect this, we will project the character onto the x and y axes, by measuring the distance to each individual side. With this we will get the probe's travelling distance from north, east, south and west until hitting the character. Further, the maximum between the minima of the first and last third is calculated. The difference of highest minimum to maximum will deal as the probe's maximum-dunked-in-distance. The direction with the highest distance wins and deals as the orientation of the open counter. Of course such a dunked-in-distance can also originate from some other constellations, but as it is only a necessary condition, the satisfying condition from the Fourier descriptors will ignore these options.

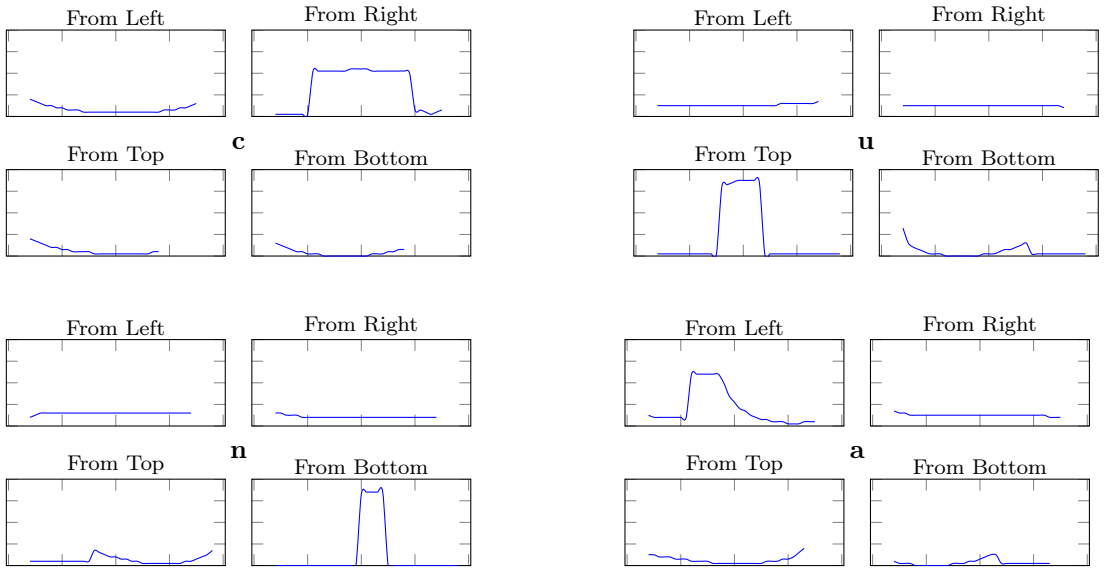


FIGURE 5.5: Arch-Detection by sending in probes from all directions. Distinct peaks are present in the direction to where the counter is open. All axes are scaled and limited equally.

All necessary orientation conditions are given in Table 5.3.

TABLE 5.3: Necessary orientation conditions for characters.

Orientation	North	East	South	West
Characters	u v H N U V	c e s C S E z Z	h m n H N	a s S z Z

Notice that [s S] and 'N' have multiple necessary conditions of which at least one must be fulfilled.

5.2.5 Necessary Condition – Stems

While the problem of [c u n a] and [s N] is solved, [b d p q] is still confusing and not detectable via the previous method. We will therefore try to detect the location of the stem, to be precise, the location where it is poking out of the bulge.

To do so the centroid of the hole will split the character into four regions. The region with the highest weight wins, since it will most likely contain the stem. The weight is just calculated as the sum of all pixels in this region. This leads to the necessary conditions for some characters depicted in Table 5.4.

Table 5.4: Necessary weight conditions for characters.

Weight	Top Right	Top Left	Bottom Left	Bottom Right
Characters	d	b	p P	q

5.2.6 Necessary Condition – Holes

Holes are a key feature of characters and a very strong necessary condition. In typographical terms they are also referred as “closed counter”, “closed apertures” or eyes.

Also, they are constant among many font-faces. The only exception might be the ’g’, where the lower eye might sometimes be open.

The number of holes therefore categorizes the characters into three groups as in Table 5.5.

Table 5.5: Necessary hole conditions for characters.

Holes	0	1	2
Characters	cfhijklmnrstuvwxyz CEFGHIJKLMNSTUVWXYZ	abeopq ADOPQR	g B

While detecting the number of holes, it is required that two holes must lie above each other and only a minimal vertical overlap of the bounding rectangle is allowed. This prevents that adjacent characters are merged that would form a rotated or mirrored ’B’ on accident. This is implemented by comparing the area of the holes inside the overlapping bounding rectangle. If this area contains more than 20% of the total area of the holes, the character is ignored.

Furthermore, it is possible that two adjacent characters are merged and form new, illegal holes. As one can see from Figure 5.6 the merged ’e’ and ’s’ form a merged construct containing two holes which outline is similar to a ’B’. Fortunately, such holes are very wide, since they are formed by two characters, which is why we will further ignore very wide holes with a bounding rectangle wider than 35 pixels.

Figure 5.6: Adjacent characters might form an intermediate hole, if connected at two points. This hole must be ignored in the filling-process during segmentation, as well as during classification.

5.2.7 Classification Engine

The final classification engine will consist of three stages: First the pre-processing for removing left over artefacts, followed by the pre-classification via necessary conditions and finally the actual classification via Fourier descriptors.

The classification will be done with a simple 1-nearest neighbour classifier, selecting the nearest neighbour from a pre-rendered reference set of characters. For finding the nearest neighbour the candidate is compared to all reference characters. If the candidate matches the necessary conditions for this reference character, the sum of differences in the Fourier descriptors is used as the measurement. A failing on matching the necessary conditions leads to a measurement of ∞. The argmin of these measurements is then used as the nearest neighbour.

Furthermore, since the arch detection alone is a pretty strong classification, a correct classification via Fourier descriptors as an arch character will lead to a boost of its difference measure by 0.5. It would be an interesting research topic, to see if the classification based on projections alone delivers satisfying results.

Moreover, i-dots are a very high indication for an 'i' or 'j'. Therefore, once the nearest neighbour has an almost straight outline like [rfijlIJ], the character is forced to be an 'i' (or j) and the measure is boosted by 0.25. I.e. the algorithm is almost forced to choose this character during backtracking. On the other hand classifications as 'i' or 'j' without an i-dot are punished by a doubling of the measurement. This has the effect that upon selecting the nearest neighbour a 'l' is favoured above an 'i' if no i-dot is present.

Another approach utilizing detected i-dots was to already consider it during the nearest neighbour search, i.e. automatically boosting the difference measure of the 'i' and 'j' if an i-dot was detected. However, this approach led to confusing results, since an i-dot candidate is equally likely to be an adjacent character shard. The boosting then led to the effect that nearly everything containing an i-dot-candidate was classified as an 'i'.

The engine is given as pseudo code in listing 5.1.

Listing 5.1: Character Classification Algorithm

```
function classifySegment(segment, reference){

  [segment, iPossible] ← cleanSegment(segment);

  [h,w] ← size(segment);

  //if segment too small or empty
  if(h<28 ∨ isEmpty(segment)){

    //could be a cut off serif
    if(segment lies at beginning ∧ cutOnlyOnce ∧ original-height<15)
      return 'ignore';
    }

    //wrong
    return 'impossible';
  }

  holes ← getNumberOfHoles(segment);
  orientation ← getOrientation(segment);
  fd ← calculateFourierDescriptors(segment);
```

```
  confidence ← inf;
  foreach(ref in reference){
    //check every reference character for possible match

    //necessary conditions
    //holes
    if(ref.holes ≠ holes)
      continue;

    //cuts - tripple splits
    if(connections ≥ 3 ∧ ref is not in 'm M K w W x T R X F E B')
      continue;

    //cuts - don't split
    if(connections > 1 ∧ ref is in 'i I j l I')
      continue;

    //orientation - require orientation match on some characters
    // ← 'only check against ref if orientation fits'
    arch_c ← 'c e s C S E z Z 2 5 7';
    arch_n ← 'h m n H N';
    arch_u ← 'u v H N U V';
    arch_a ← 's S z Z a 3 2 5 7';
    if(orientation ≠ 'c' ∧ ref is in arch_c)
      continue;
    //analogue una

    //weights - require weight match on some characters
    if(ref = 'd' ∧ orientation ≠ 'topRight')
      continue;
    //analogue for [bpPq]

    //Phase-Check
    noPhaseCheck ← 'oODiIljAHcCmnhuvUVdbpPq1235780';
    if(ref is not in noPhaseCheck){
      if(fd.phase differs too much from ref.phase)
        continue;
    }

    //necessary conditions passed. compare fourier descriptors.
    results[] ← sum( abs(fd-ref.fd) );

    //classification as 'i' or 'j' without i-dot is punished
    if(¬ iPossible ∧ (ref = 'i' ∨ ref = 'j'))
      result *← 1.5;
  }

  //1NN
  NN ← min(results);
  char ← NN.char;
  confidence ← NN.result;

  //force i if i-dot was found
  if(iPossible ∧ char like 'r f t i l I')
    char ← 'i'; confidence *← 0.25;
  if(iPossible ∧ char like 'j J')
    char ← 'j';  confidence *← 0.25;

  //boost based on orientation
  //the orientation itself is a very strong indicator
  if(orientation check performed and matched)
    confidence *← 0.5;

  return [char, confidence];
}
```

5.3 Evaluation

5.3.1 Self-Distorted

For a first evaluation of the proposed algorithm, the engine is given a rendered version of the font "Times New Roman". All 52 characters are rendered and given as reference, completely untouched.

For testing the same set is taken, but distorted. Each character is rotated by $\pm 5°$ and $\pm 10°$, scaled unproportionally to a fixed size of 48x48 Pixels and further distorted by a vertical and horizontal sin/cos wave. The wave varied between an amplitude of ± 3 in steps of 0.5. This resulted in a test set of 3380 characters, we will call this test set the "distorted times" test set. Figure 5.7 shows some examples presented to the algorithm in comparison to what it knows about these characters.

(A) Examples from the distorted times test set.

(B) Corresponding reference characters given to the algorithm.

FIGURE 5.7: Comparison of what the engine is confronted in the distorted times test set with and what it knows.

For comparison the same set is presented to a state of the art OCR engine, called "Tesseract". Wikipedia [Wik] quotes *"Tesseract is considered one of the most accurate open source OCR engines currently available."*

For fairness reasons Tesseract is configured as only detecting [a-zA-Z] and put into "Treat the image as a single character" mode.

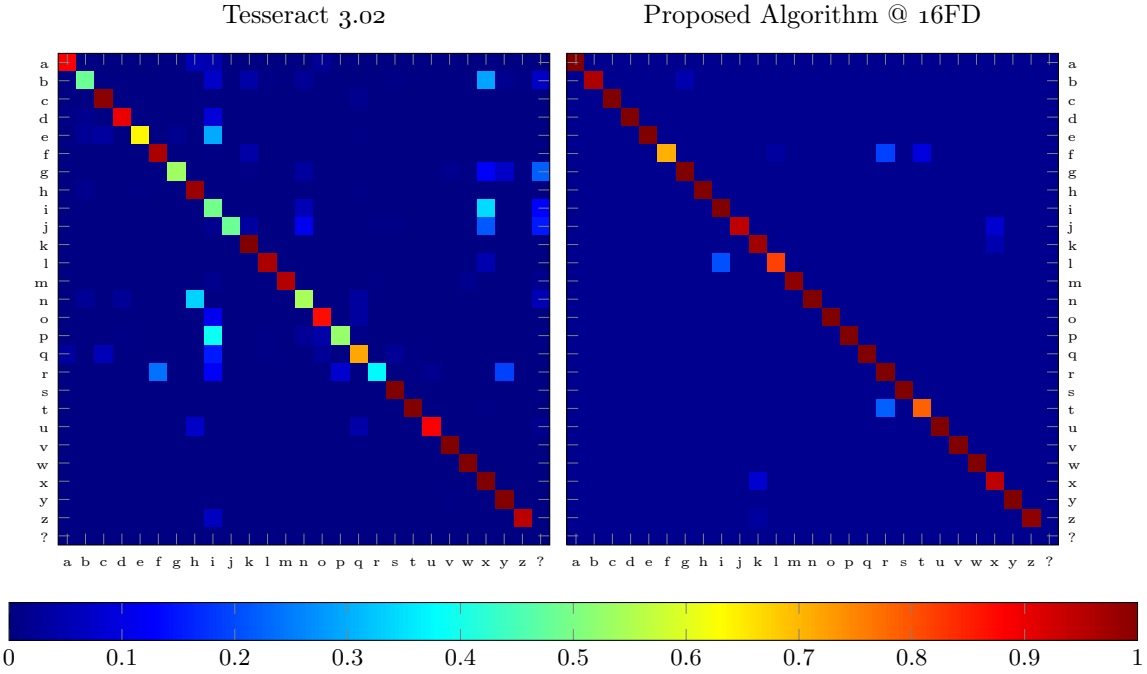


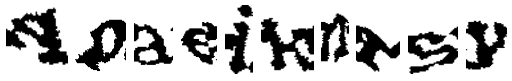
FIGURE 5.8: Comparison of confusion matrices for the generated distorted times test set. Y-axis indicates the target, x-axis indicates the classification result. The best result would be a straight, red line. Both perform really well. The proposed algorithm however still performs slightly better than Tesseract with a mean recognition rate of 95.95% against 80.68%. Note that the algorithm only has knowledge about one single, undistorted instance of each character!

Figure 5.8 shows the confusion matrices of Tesseract and our algorithm. As one can see both perform pretty well, still our algorithm performed with an average recognition rate of 96% better than Tesseract with 80%. A remarkable achievement, considering it only knows one instance of every individual character. All confusion matrices can also be found in detail in the appendix B on page 59 et seqq.

5.3.2 reCaptcha Characters

We will now evaluate the performance on the manually segmented characters obtained from the real reCaptchas. As mentioned earlier broken or wrong connected characters are not part of this thesis. Therefore, 315 characters were excluded from the 2468 test characters.

Since these characters are even more distorted than the previous test set, we will grant our engine also a $\pm 5°$ and $\pm 10°$ rotated instance of each character. Also, the reference set does not consist of one single font-family, but is selected from Times New Roman, Cambria and Calibri. It is selected by best (human) guess providing the best discrimination ability. Examples for the captcha test set and the reference characters are given in Figure 5.9



(A) Examples from the captcha test set.

(B) Corresponding reference characters given to the algorithm.

FIGURE 5.9: Comparison of what the engine is confronted with in the captcha test set and what it knows.

Moreover, a very large amount of 's' characters showed up a connection of the upper serif and the spine. Therefore, a manually connected 's' from Times New Roman is given to the engine.

Also the letter 'r' is given in two versions, one having a very dominant serif drop (**r**) and one with a sans-serif drop (**r**). All in all this lead to a reference set of 275 instances.

This lead to the following remarkable results. Figure 5.10 shows the confusion matrices of our proposed algorithm and the results of **tesseract**. With a recognition rate of 78.7% our engine outshines Tesseract with just 39%. Keep in mind that it only has knowledge about sharp, undistorted and clean instances of every single character. Again Tesseract was configured to only detect letters and put into "Treat the image as a single character" mode.

Considering the confusion matrix 5.10 one can see that a significant amount of 'i' and 'j' were confused. A problem that should be easily solvable by postprocessing. Moreover, it is questionable whether it is in fact even recognizable by a human. Also a rather large amount of 'f' were confused as 't' and vice versa. While being understandable, this problem should be again solvable by post-processing.

One might notice that no 'w' are present. Sadly no 'w' got into the test set, since it is a very rare character (see Section 7.1) and was always connected wrong or completely broken.

Interesting is the amount of about 30% from 'o' confused as 'D' (big and small letters are consolidated in the given graphic). This is due to the fact that a side-gated 'o' generates a straight line at the edge, followed by the normal loop of the 'o'. The generated character then just looks like a 'D'. Of course this could be discriminated by the dimensions of the character. However, big and small letters are often squeezed and scaled in the captchas.

The scope and possibilities of post-processing will be discussed in chapter 7.1.

For another evaluation the proposed algorithm was compared to the "Abbyy **Finereader** Engine 11", for which Abbyy gratefully supplied a trial license. The results are presented in Figure 5.11. For evaluation Finereader was again only allowed to use letters and put into "Text-Extraction Accuracy", the slowest, but most accurate processing mode. As we can see from the confusion

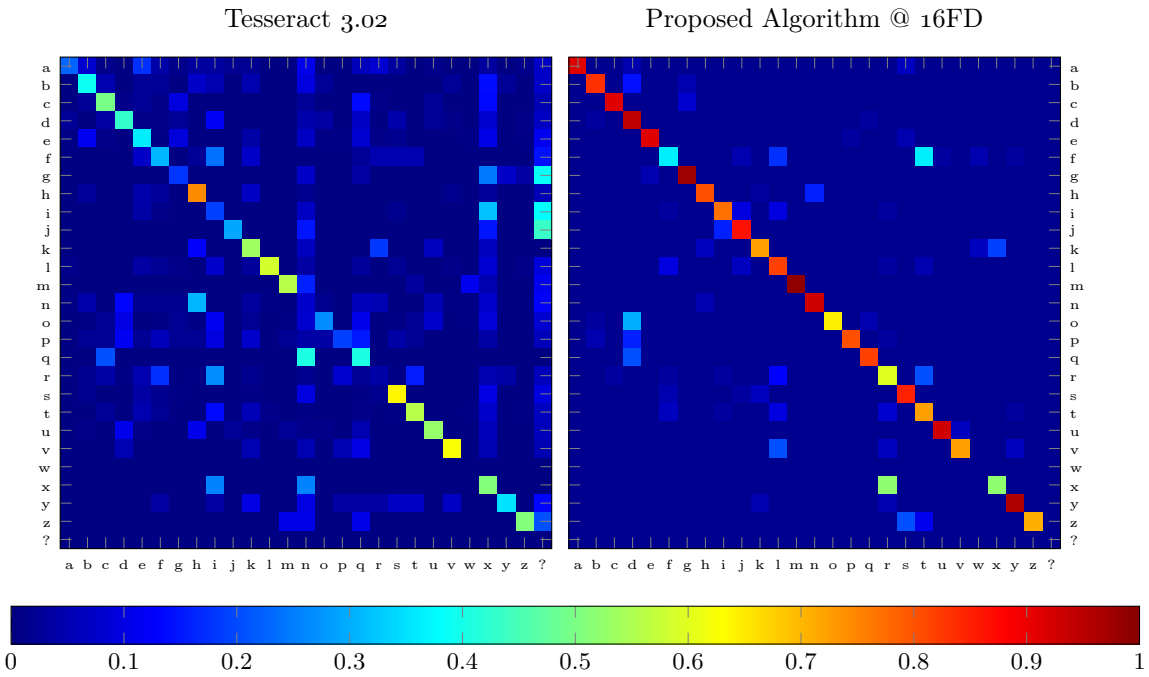


FIGURE 5.10: Comparison of confusion matrices for the captcha test set. The straight red line, is nearly achieved by the proposed algorithm with a mean recognition rate of 79%.

matrix Finereader does not perform very well. Admittedly, we want to point out that Finereader has a huge amount of tweaking parameters and is very powerful in processing even handwritten documents or badly printed recipes. Yet, it seems that Finereader is not able to cope with the low resolution of the characters presented to it. Frankly it could be possible that Finereader was configured wrong; notwithstanding equal efforts were taken to configure Tesseract and Finereader.

Finally, we will benchmark the proposed algorithm against itself with a different amount of Fourier descriptors. Figure 5.12 shows both confusion matrices using 16 and 32 Fourier descriptors. As one can see both confusion matrices are nearly identical. In fact, the overall increase in accuracy is only 0.16%. Indeed, the complete algorithm will perform slightly better with 32 Fourier descriptors. It will be able to discard more wrong characters as it checks the outline for more details.

The small difference should be explainable by two characteristics. For once, the magnitude of feature reduction from about 2500 features (from a 50x50 logical image) down to 16 or 32 features is so high, that the scale between 16 and 32 features is irrelevant. Secondly the Fourier descriptors are naturally weighted—the higher the frequency, the smaller the coefficient. With that the difference of high frequency coefficients contributes less to the complete sum of differences.

Having developed a remarkable accurate classification engine dwarfing the “most accurate OCR system”, we are now able to process the complete text segments.

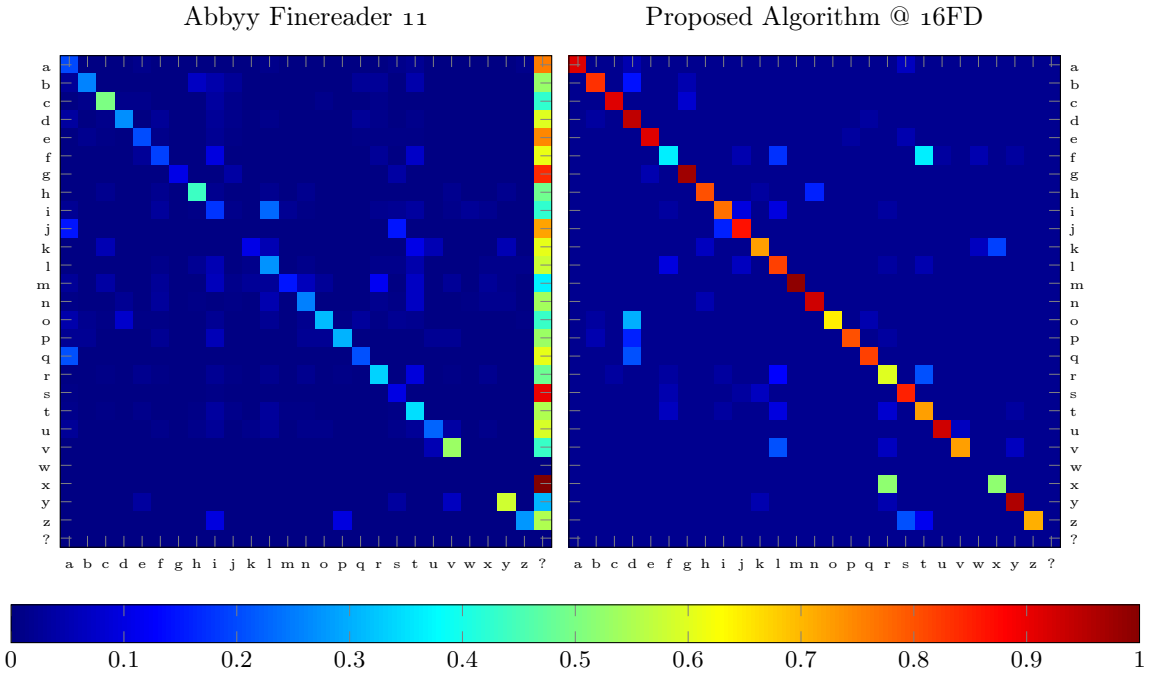


FIGURE 5.11: Comparison of confusion matrices. The commercially available Abbyy Finereader performs poorly against the proposed algorithm. Admittedly, one has to notice that Finereader is not primarily designed for single character recognition and could be heavily trained to recognize these characters.

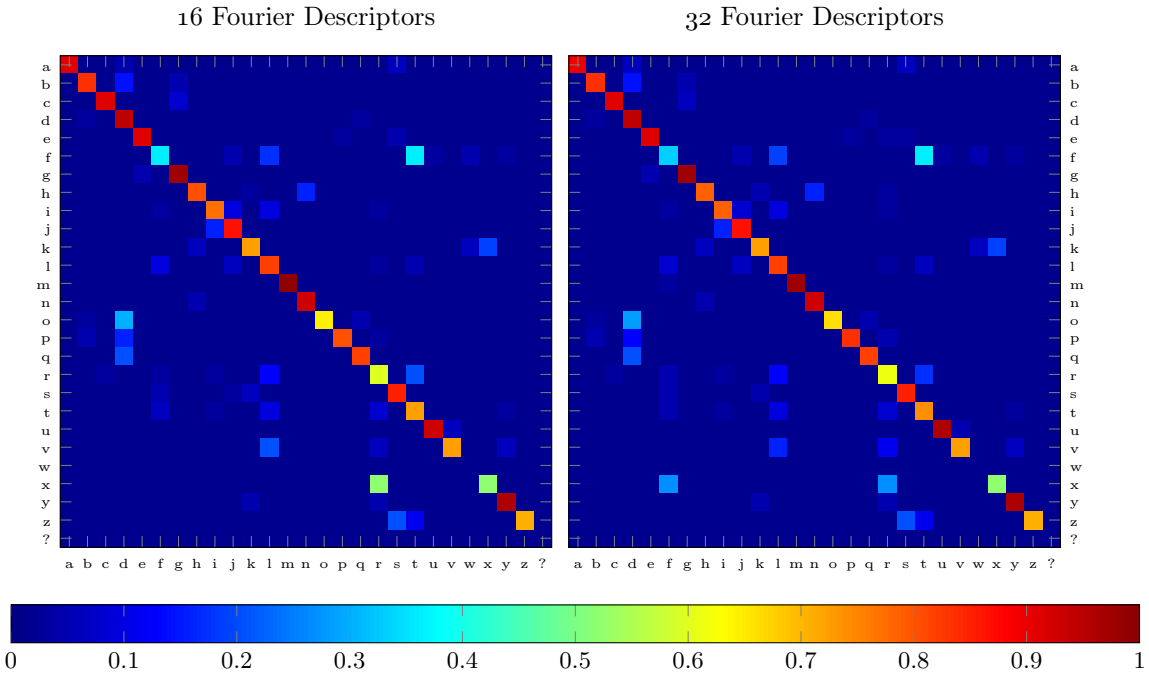


FIGURE 5.12: Comparison of confusion matrices for the proposed algorithm with 16 and 32 Fourier descriptors. Nearly no difference is visible, still the algorithm will perform better with 32 Fourier descriptors, since it will more likely disregard false-characters.

Chapter 6

Backtracking and Evaluation

With the developed classification system we are now able to combine all previously developed steps, and hopefully be able to correctly detect the distorted text segments.

6.1 Viterbi

As mentioned earlier the idea was to oversegment the image and then merge adjacent segments. Some merged combinations will then be classified as the correct character with a low difference measure. Other wrong combinations will be discarded by the engine or be classified with a high difference measure. For this purpose a classic Viterbi algorithm is used.

Of course a once used segment is not allowed to be used again in a following combination, i.e. it does not greedily select the lowest difference measures, but aims that the overall picture is satisfying.

Figure 6.1 displays the process at the previously seen "nnguit"-captcha. It is chosen, because it shows important properties of the Viterbi algorithm and is rather short.

At first the algorithm merges and classifies up to three adjacent segments, from which the depicted graph in Figure 6.1(top) is formed. Further, all incoming connections into one segment are checked and the lowest accumulated difference is chosen. I.e. the segment's classification measure (denoted along the path) is added to the origin's measure of this path. From these accumulated difference measures the lowest is chosen and marked as this node's measure. With this the second graph 6.1(bottom) is formed.

At the end the best route is backtracked by following the marked decisions.

This process can also be modelled as a hidden Markov model (HMM). The (merged) segments form observed states originating from hidden states—the real characters. Using the Viterbi algorithm we determine the most likely sequence of these hidden states, i.e. the underlying characters from which the single segments originated, which then solves the captcha. For a detail and indepth analysis of this approach one can consider [Vin02] or various works of Michel Gilloux on cursive script recognition.

6.2 Further Tweaks

During the classification and Viterbi some additional tweaks are added to the classification engine.

For once it might be possible that a triple connection leads to some arbitrary construct, whose outline accidentally resembles some character. Since it can jump over three single connections,

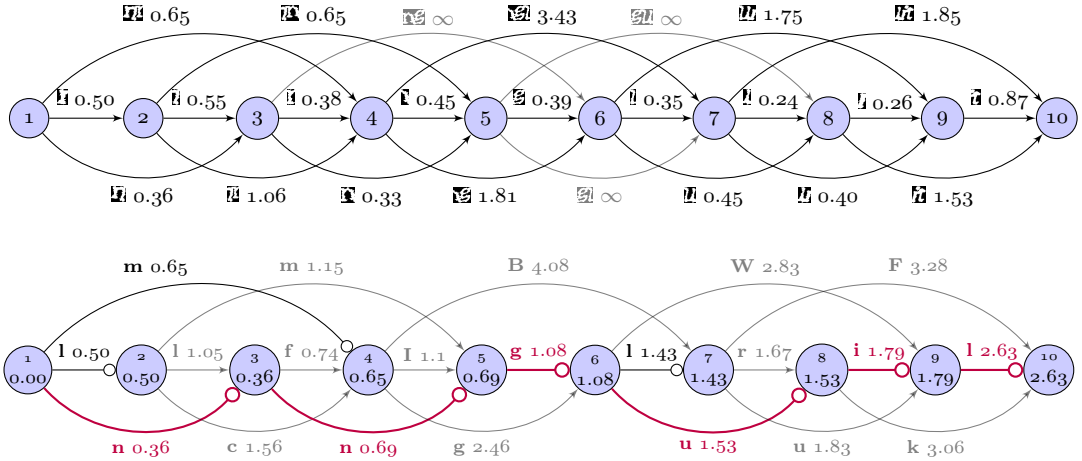


FIGURE 6.1: Viterbi Algorithm for the "nnguit"-Captcha. Top: The initial errors for every single and merged segment. Bottom: The accumulated errors and classifications. The best incoming connections are marked with —○. The optimal backtracked path is marked in purple.
Notice that it is not greedy (i.e. always selects the lowest possible measurement). Otherwise it would pick the 0.35 between 6 to 7, instead of the worse, but correct 0.45 from 6 to 8. Also, it is not forward orientated (i.e. evaluating all possible routes to where the segment points), where it would select the 0.65 at the beginning, with which it could directly jump to segment 4, to where the next best possible way would be $0.36 + 0.38 = 0.74$. It is backward orientated, such that the best way is chosen that minimizes the total accumulated error along the path.

the Viterbi would heartily choose this way, even if it has a rather high difference measure, which normally should indicate a wrong decision.

On the other hand some characters cannot be formed by multiple connections. For example an 'i' would never be segmented into multiple segments. We therefore define three more necessary conditions for merged segments:

The segment is not allowed to be wider than the **maximum letter width**. This is most likely a 'M' or 'W'. Empirical studies showed that a threshold of 110 Pixels showed rather good results. This also prevents that stretched afar segments are merged; exactly what we intended.

A **triple connection** is only allowed on special characters. A triple split is common for capital letters, like an F. The three stages of stem-end, middle bar-end and top bar-end split the character into three parts. Also 'm' and 'w' are always split into three parts, as their counters form minima. We allow a triple split on: [m M K w W x T R X F E B].

Finally, some characters are only allowed to originate from a **single segment**, i.e. they can't origin from merged segments. This includes small, solid characters as: [i I j l I]. This is important, because some arbitrarily merged characters might always form a "stem"-like outline, leading to a rather good classification as an 'l' or similar.

Furthermore, since the algorithm can skip one or two segments when taking a merged object, the difference measure can be twice as high as a single element. This would lead to many **large jumps**, ignoring valuable single segments. Therefore, multi-connections are weighted globally by a factor of 1.5.

Figure 6.2 shows the algorithm finally solving the "Bblicie"-captcha, which accompanied us since the beginning.

FIGURE 6.2: Final backtracked cutting positions for the 'Bblicie'-captcha with the classifications and thus solved captcha.

6.3 Evaluation

We are now able to completely evaluate the proposed algorithm. For evaluation the acquired captcha test set of roughly 400 captchas is used. As neither the classification engine nor any other part is trained with data from this test set, no cross-validation should be required, since overfitting cannot occur. From the acquired 410 captchas, only the 399 captchas are used from which the challenge was extracted correctly. This is done because basically both parts could be processed one after another, but we are only interested in one of them. Admittedly one might argue that selected captchas, on which we based some special exceptions, must be excluded from the test set; since we heavily generalized these exceptions and tried to find real world applications, we will still use them.

Since the result of the algorithm will be a text-string, we first need to find a way to quantify the result.

6.3.1 String Difference Measure

While acquiring the test data, a response was sent to the reCaptcha servers which then returned a boolean answer "accepted" or "fail". We want to point out that the answered string is not necessarily the exact string that was presented or was intended by the captcha. During the entering phase some observations have been made that lead to the conclusion that a certain amount of error is acceptable. For example at least one character was allowed to be left out. Also, some easily confusable characters like 'I/l', 'c/e' or 'i/j' were allowed to be interchanged.

Of course the underlying difference measure is not disclosed. We therefore must refer to a similar measurement. One would for example be the Damerau–Levenshtein distance, a difference measure for two strings that will give the number of operations needed to align both strings. This is also known as the "optimal alignment problem", also found in the world of computational biology in the range of DNA sequence alignment.

The Damerau–Levenshtein distance allows operations of insertion (inserting a left out character), deletion (removing one additional character), mismatch (exchanging a character) and—this is an addition to the optimal alignment problem—the flipping of two adjacent characters.

Since we already mentioned that some characters are basically similar and leave even no choice for the human for discrimination, the strings are normalized by a replacement of similar character groups. The groups of [i l I j t f r], [D o], [B a] [X x k K] and [u U v V] are treated as the same letters. [B a] is included in this set, as a significant amount of a's showed a connection of the "instroke" with the "bowl". Furthermore, the strings are compared case-insensitive and adjacent 'll' are treated as an 'u'.

In order to justify this measurement we will first perform an evaluation among humans on the test set. This is intended to also show up to which distance an answer to a captcha should be considered correct.

6.3.2 Human Performance

To evaluate the performance of humans on the test set, a website was generated and posted on facebook with the ask for help.

The website featured a simple textbox and the challenge from the captcha. The captcha was presented in the original dimensions and left unprocessed, only the challenge was extracted. Moreover, the time was measured from fully loading the image up to submitting the form. On page load the focus was automatically set into the textbox and the form could be submitted by hitting Enter; therefore the time measurement should be accurate.

The captcha was selected randomly from the test set on every page load.

The test was well received with over 1000 (again) solved captchas and over 50 participants within just one day. After one day, no further entries occurred, since it went out of focus inside the facebook timeline. We will assume that (almost) all entered solutions must be accepted by the captcha systems. Otherwise it would not fulfil its requirement from separating humans from computers. Table 6.1 shows the results of this evaluation.

TABLE 6.1: Human results for the unprocessed challenges. With only 65% complete matches, the captcha system must at least allow some range of inaccuracy. Otherwise it would nearly lock out every second human. We conclude that a Damerau–Levenshtein distance (DLD) of two is still acceptable as a full match.

DLD	**Absolute**	**Percent**
0	688	65.15%
1	219	85.89%
2	71	92.61%
3	15	94.03%
4	6	94.60%
5	6	95.17%
6	1	95.27%
7	4	95.64%
8	3	95.93%
Empty	43	100.00%
Total	1056	

As one can see only 65% of the captchas were solved with a Damerau–Levenshtein distance of 0, even after normalization. So only accepting answers with a distance of 0, i.e. an exact match, would lock out nearly half of all humans from the test. A number way too high for usability. Therefore, at least some error must be allowed. As we can see from the table, 92% of the humans achieved a Damerau–Levenshtein distance of smaller than 3. We will therefore accept answers with a distance smaller or equal to 2 as a "full or almost match". A distance of 3 or 4 will be considered as at least a "partial match". Distances above that might still have some partial hits, but will be considered as "wrong".

6.3.3 Evaluation of Tesseract and Finereader, unprocessed

Since the previous evaluation has set a very high benchmark of 92% correct recognitions, we will now evaluate how well current, available, profound systems perform in this setting.

For this the applications already used previously—Tesseract and Finereader—are used. Both applications are again configured to only detect letters. Tesseract was further put into "Treat the image as a single word."-mode. Both applications were then confronted with the unprocessed extracted challenges. Table 6.2 shows the results.

Table 6.2: Evaluation results of Abbyy Finereader 11 and Tesseract 3.02 for unprocessed challenges of the captcha test set (399 captchas).

DLD	**Finereader Pure**	**Tesseract Pure**
0	0	3
1	1	11
2	5	30
3	10	30
4	7	50
5	4	61
6	2	81
7	4	43
8	0	5
9	1	1
10	0	
11	0	
Empty	365	84
Full/Almost	6	44
Partial	17	80
No	376	275
Full/Almost	**1.50%**	**11.03%**
Partial	4.26%	20.05%
No	94.24%	68.92%

As we can see from the table, both applications perform pretty bad. Of course this is a very harsh test for the applications, since both are primarily trained for high resolution scans of sharp texts. Most likely the low resolution in combination with the sloped baseline and shearing of the letters confuses both systems. However, at least the shearing should not confuse the system, since it is natural for *italic* text.

6.3.4 Evaluation of Tesseract and Finereader, processed

To be fair we will also repeat the previous test, with preprocessed challenges. We therefore deslope and deslant all segments and present them to the applications again.

This is fair since the wavy slope is an especially rare occurrence (but still possible and reasonable) and both applications are clearly not trained to it.

The results (Table 6.3) show a much higher recognition rate, but still half of the words are detected with a Damerau–Levenshtein distance above 4. In fact, many words were just ignored completely. Since an (almost) empty answer leads to a Damerau–Levenshtein distance of the reference string length, answers with zero or one characters are marked as "empty" in the table. Such empty words are a peculiarity that our algorithm does not do, as it will always output *something*, even if it is useless. It is questionable whether this is better or not. We will say that answering gibberish, instead of an empty or one-character word is better, since it will give the reader a hint that the OCR system was not able to recognize something. A left out word inside a text might not be that obvious. Furthermore, every spell-check system would mark the gibberish.

With Tesseract's performance of about 25% we now have a target to compare our algorithm against.

TABLE 6.3: Evaluation results of Finereader and Tesseract for the preprocessed (desloped and deslanted) challenges of the captcha test set (399 captchas). One can clearly see that neither Tesseract nor Finereader is prepared for such distorted text segments.

DLD	**Finereader**	**Tesseract**	**Finereader Pure**	**Tesseract Pure**
0	8	27	0	3
1	8	22	1	11
2	27	51	5	30
3	36	51	10	30
4	26	59	7	50
5	20	54	4	61
6	19	51	2	81
7	3	26	4	43
8	1	5	0	5
9	1	0	1	1
10		1		
11				
Empty	250	52	365	84
Full/Almost	43	100	6	44
Partial	62	110	17	80
No	294	189	376	275
Full/Almost	**10.78%**	**25.06%**	**1.50%**	**11.03%**
Partial	15.54%	27.57%	4.26%	20.05%
No	73.68%	47.37%	94.24%	68.92%

6.3.5 Evaluation of the Proposed Algorithm

With the set target for our algorithm, we will now evaluate the performance of our proposed algorithm. We will further compare the performance of utilizing 16 and 32 Fourier descriptors. Again all 399 captchas of the test set are processed by the algorithm. Table 6.4 shows the results in comparison to Tesseract and Finereader.

With a recognition rate of 29.5% we have beaten the set target and the performance of Tesseract and Finereader. We can further note that the proposed algorithm generates much more partial matches than the competitors. A fact that might come in handy in postprocessing of real world texts, since a potential spell-check might still be able to reconstruct the original word.

While the 29.5% are a remarkable, pleasant result and it performs better—especially in the partial matches—than Tesseract it still performs below its potential. We have previously seen that the classification engine itself is capable of detecting 79% of the characters. With less than the half of this in the final results, it should be capable of more. The results of Tesseract range in the same dimension. With a final result of 25% and its capabilities of detecting 39% of the characters the results are reasonable. We can conclude that the weakness of the proposed algorithm is its segmentation algorithm. Tesseract implements are more sophisticated approach utilizing a box-scanning. Still, even our simple approach was able to outperform it slightly.

We can further see that the difference between 16 and 32 Fourier descriptors is marginal. With just one more exact match and a slight shift in the distances, the results are nearly the same. Since a difference in processing times was not measurable, we will stick to the 32 Fourier descriptors.

Figure 6.3 shows some selected solved examples from the test set.

TABLE 6.4: Final evaluation results of the proposed algorithm utilizing 16 and 32 Fourier descriptors. For comparison the results of Tesseract, Finereader and the humans are given. With a slightly higher accuracy in full/almost matches and a much higher partial hits our algorithm performs really well. Notice that Tesseract completely gives up on 13% of the captchas, while our algorithm still tries to at least find some matches.

DLD	**16 FD**	**32 FD**	**Tesseract**	**Finereader**	**Humans**
0	21	22	27	8	688
1	34	36	22	8	219
2	62	60	51	27	71
3	74	87	51	36	15
4	89	76	59	26	6
5	59	66	54	20	6
6	46	35	51	19	1
7	12	14	26	3	4
8	2	3	5	1	3
9			0	1	
10			1		
11					
Empty			52	250	43
Full/Almost	117	118	100	43	978
Partial	163	163	110	62	21
No	119	118	189	294	57
Full/Almost	**29.32%**	**29.57%**	**25.06%**	**10.78%**	**92.61%**
Partial	40.85%	40.85%	27.57%	15.54%	1.99%
No	29.82%	29.57%	47.37%	73.68%	5.40%

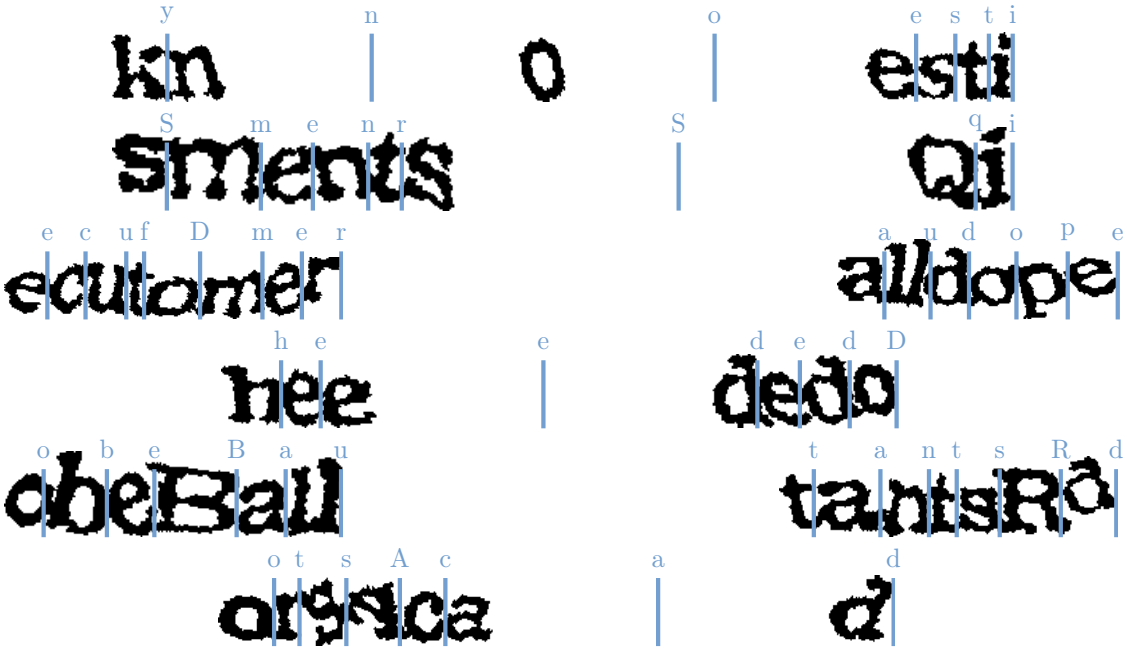


FIGURE 6.3: Selected, processed and solved examples from the captcha test set.

Time Requirement As mentioned earlier, the human performance was also evaluated with regard to solving time. The time taken from fully loading the image up to submitting the answer was measured.

Of course the time requirements differ from human to human, depending on the typing speed and how well the human is used to captchas in general. Since the testpage was published using facebook, we can assume that all 52 users should be familiar with captchas in general and further be kind of used to typing. In other words, the resulting times will set a higher benchmark than the average.

For evaluating the mean time required to process a captcha we will eliminate outliers from the answers. In some cases the time measurement failed and a time of zero was submitted. Since the time is measured using JavaScript, certain browser-extensions might have caused this. Further, times above 30s were ignored. These times might origin from a case where the human did not process the captcha immediately after loading the page, e.g. the user has left the computer or was distracted from the page.

As said earlier the time measurement was started right after the challenge-image loaded completely. Another starting point would have been the time, when the user entered the first character. However, this would not reflect the complete process, since most of the humans will first start typing, after fully processing the complete captcha (i.e. reading it). This would have measured the *typing speed* of each user and not the time required to *solve* the captcha.

Considering this, a human took 6.64s on average to process a captcha with a mean deviation of 3.66s. In comparison our algorithm just needs 1.33s[1] per captcha with a mean deviation of 0.11s.

Considering that a human recognizes 92.6% of all captchas but takes this long, it is reasonable to consider the throughput of the algorithm: Our algorithm achieves 13.3captchas/minute, whereas a human is just able to process 8.37captchas/minute. In other words, one could say that the algorithm achieves a recognition rate of 147% in comparison to a human.

With this we propose that the algorithm is able to beat the reCaptcha system. A big claim we will try to verify in the next section.

6.3.6 Live Test / Verification Set

In order to verify the previous results, the idea was to perform a live test of the algorithm against the reCaptcha system.

To connect Matlab with the reCaptcha system, a small backend in PHP was developed, since no direct API implementation is available for Matlab. Matlab first requests a new captcha from the PHP script. The script then fetches a new captcha, downloads the image and returns the filename. After processing the file, Matlab calls the script again with the processed result which is then verified by the reCaptcha servers.

Based on experience the current reCaptcha implementation was constant since at least spring 2012. Unfortunately and strictly according to Murphy's Law reCaptcha changed its system right after finishing the research on this system.

Figure 6.4 shows two examples of this new version. It appears that Google is now using the captcha system for increasing the accuracy of Google Maps. It seems to present a house number taken from Google StreetView in combination with a new challenge consisting solely of numbers. The fact that only numbers are present increases the usability of the system, since switching from entering numbers to characters is inconvenient. The system remains basically the same, with the StreetView image as pro-bono part and the generated numbers as challenge. Again only the challenge has to be entered. Although this time the number must be entered exactly, no mistakes are allowed

[1] The benchmarks were performed on an Intel Core i5-750, 4x2.67GHz, 16GB DDR3-1600, Windows 7 Professional 64Bit and Matlab R2013b.

anymore. Upon entering more than four of these number-captchas or a false answer the system refers to the old versions.

This is a feature also mentioned in the beginning implemented by "solveMedia". If the system detects a possible robot it hardens itself, while a human is presented with easily solvable captchas. In order to disable mass-inputs by humans (services exist that solve/break captchas by employing low paid humans), the system hardens itself after four entered captchas.

Upon hardening the system returns to the old system at first glance. Sadly a live test against this performed very poorly. Only an acceptance rate of about 4% was achieved. This might have several reasons, which will now be discussed.

FIGURE 6.4: New version of the reCaptcha system as of fall 2013.

At first, as we can see from Figure 6.5, the captchas appear to be much more dense. With a much smaller letter spacing, generating cutting-hypotheses becomes incredibly hard. As one can see from the "mtetai"-captcha, a classic caliper-distance would fail completely. But even our projection-based approach would fail. The extreme tight fitting of 'm' and 't' or 'a' and 'i' sadly no longer enables us to cut at these positions. It is even not clear whether it is an 'nn' or an 'm'.

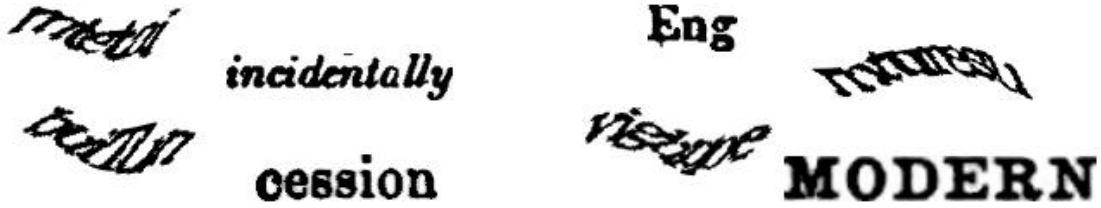


FIGURE 6.5: The new, hardened version of the old system as of fall 2013.

This is a really intelligent approach of the reCaptcha system. With the number-challenges being very easy for a human, but very hard for a robot (see Section 7.2), humans get very easy challenges. Upon a wrong input, a very hard challenge is presented that even further excludes robots. Yet, humans are still able to more or less recognize even these hard versions. Upon solving a hard challenge the system can again be sure to be talking to a human, and can return to easier ones, which are again easy solvable by a human. This is a win-win situation, since humans are able to solve them easily and robots are excluded.

Furthermore, we have to question, whether our first data acquisition of the captcha test set might have been error prone. It might be possible that this "harden upon potential robot" was already present in the old version. Because the data was acquired by a human, the system believed to be talking to a human, and therefore presented easier captchas. Once the algorithm tries to solve these captchas (according to the test set about every third captcha is detected correctly), already two captchas were entered incorrectly. Since that, the system might already have hardened and therefore present much harder captchas, as it has seen in the test set.

Nevertheless, since the goal was to develop a very robust recognition system and not primarily breaking the reCaptcha system, we still achieved our goal. However, one could argue heretically that our algorithm fails on the live-test, only because all defined thresholds (even though we tried to keep absolute threshold down) were tightly fitted to the test set.

In order to prove these heretical voices wrong and verify our results, we want to run the test against a new, completely unseen, verification set. Since the system is not available anymore, freshly gathering this data was not possible. Fortunately, a local download manager kept a local cache

of entered captchas. From this cache 335 captchas were collectable, which were entered during a timespan from December 2012 up to May 2013. The captchas appear to be of the same version as our test set.

Again all 335 captchas were solved manually and then compared to the outcome of the algorithm. Still, one captcha had to be excluded since it was not readable for a human. Table 6.5 shows the results for the verification set, which match the results for the test set.

TABLE 6.5: Final verification results. With the verification set delivering nearly the same results, we can say that our algorithm works very well. For comparison the results of Tesseract and the humans of the test set are given.

DLD	**Test Set**	**Verification Set**	**Tesseract**	**Humans**
0	22	15	27	688
1	36	30	22	219
2	60	53	51	71
3	87	64	51	15
4	76	73	59	6
5	66	53	54	6
6	35	30	51	1
7	14	13	26	4
8	3	2	5	3
9		1	0	
10			1	
11				
Empty			52	43
Total	399	334	399	1056
Full/Almost	118	98	100	978
Partial	163	137	110	21
No	118	99	189	57
Full/Almost	**29.57%**	**29.34%**	**25.06%**	**92.61%**
Partial	40.85%	41.02%	27.57%	1.99%
No	29.57%	29.64%	47.37%	5.40%

Chapter 7

Outlook and Conclusion

We will dedicate this chapter to an outlook on what could further be implemented in the form of postprocessing. We will show why postprocessing is possible and what possibilities it could hold.

Furthermore, we will give an conclusion to our algorithms, show its adaptability and discuss its limits and potentials.

7.1 Postprocessing

We will begin with a stochastic analysis of the captchas. While manually acquiring the datasets the feeling arose that the captchas are not generated completely randomly. In a sense they were always pronounceable, and sometimes even spelled some kind of word. As an example one could take the "knoesti" captcha from the beginning, which sounds like a German word. On top of being pronounceable they were always convenient to type.

This raised the assumption that the captchas underlie a probabilistic generation model, which is also present in a human language. All human languages underlie a pattern of letter frequencies and bigram frequencies. Bigrams, or rather n-grams, are an alignment of n consecutive characters, e.g. very frequent bigrams (n=2) are "in" in English or "en" in German.

We therefore analyse the letter frequencies in all entered captchas, all captchas from the test and verification set. While 735 unique words are not very sufficient for generating a reliable stochastic analysis, it still shows that certain characters are favoured among others. For comparison the freely available books "The Adventures of Sherlock Holmes" by Arthur Conan Doyle, "Romeo and Juliet" by William Shakespeare and "Grimms Fairy Tales" by the Brothers Grimm were analysed. All books were obtained from the Gutenberg Project [Gut]. Prior to generating the frequencies, all non-letter characters were erased and the complete text was converted into lowercase. This lead to the statistics in Figure 7.1.

As we can see the captchas clearly follow a trend comparable to the English language. This helps either reading the captcha, as well as typing the captcha. On the other hand, this fact can also help us and is very common in text recognition as part of postprocessing. The different characters can help us on deciding which nearest neighbour to choose, since some are more likely to occur. As an example one can consider a 'k' and an 'x'. While both—in particular when distorted—are very similar, an 'x' is with just 0.37% much less likely to occur compared to a 'k' with a letter frequency of 1.4%.

An implementation option for this could be to consider the n (e.g. 3) nearest characters, weigh their difference measure by their probability and then choose the lowest. After the selection the unaltered measure should be used, to maintain comparability during the Viterbi algorithm. Notice that we talk about nearest *characters* and not neighbours, since in our implementation a character

is presented with multiple instances. Considering just e.g. three nearest neighbours would very often bring up three times the same character. It should therefore be better to consider the n nearest characters and choose the lowest difference measures of those for the group.

Another option would be to again consider the n nearest characters and then simply choosing the most probable. However, in this case, an 'x' would almost never be chosen over an 'k', even though the difference measure is much higher.

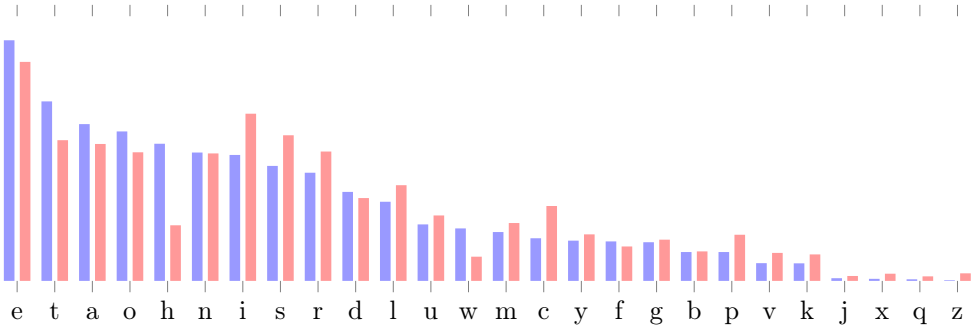


FIGURE 7.1: Letter frequencies for English (Blue) compared to the letter frequencies from all captchas (Red). While not being completely identical, a clear trend is visible.

As mentioned before, the other statistic is the bigram frequency. For this again all 735 unique words are chosen and a dictionary is built (in the case of the captchas this of course leads to a 735 word dictionary, since the captchas are all unique). The same is done for the mentioned books, which leads to a dictionary consisting of 11335 words. Of all these distinct words the bigram frequencies were analysed and are displayed in Figure 7.2. One could of course also analyse all tri- and four-grams.

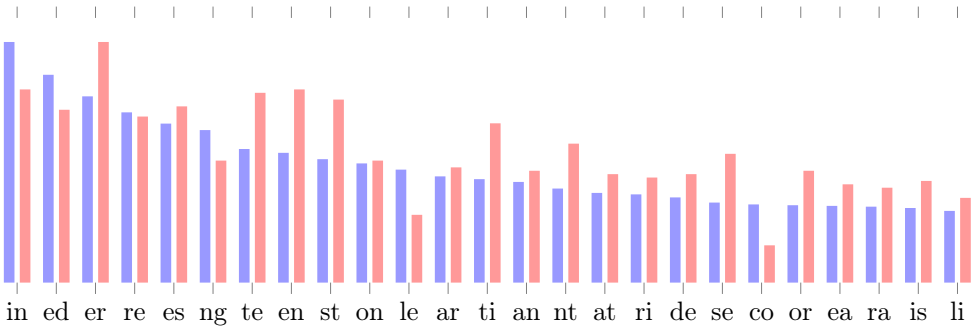


FIGURE 7.2: The top 25 bigrams from the generated English dictionary (Blue) compared to the corresponding frequencies of all captchas (Red). The frequencies are normalized to the highest occurrence. Almost all English bigrams are also dominant in the captchas. Of the top 10 English bigrams, 7 are also found in the top 10 captcha bigrams.

It is evident from this statistic that also a trend is visible. Admittedly, 735 unique words are again too sparse to create a reliable statistic. Still, we can see that certain bigrams are much more likely than others. Noticeable were also that really unusual—or not existing at all—bigrams like "xw" or "yx" from the English language were also not present in the bigrams of the captchas. Again this helps the captcha in being more typeable, since humans are more trained to certain character sequences than others. Again this can also be utilized in postprocessing.

During the Viterbi algorithm certain paths can be favoured in regard to others. As one implementation one could check the bigram formed by the current path and the character which lead to the

local optimum of the origin node. Another option would be to accept multiple paths through the Viterbi diagram and afterwards choose the path with the most likely bigrams.

The last postprocessing option is a classic spell-checking system. Of course this only works with a profound dictionary and an underlying real language. Spell checking reCaptchas would be of no avail, since they are only on "accident" real words. With other captcha systems (e.g. solveMedia) this approach should lead to great results. On top of that, a spell-checking system would greatly benefit from the proposed algorithm, since our algorithm always generates *some* answer, instead of completely giving up or stopping once the recognition becomes unreliable. Also, it generates much more partial matches, from which a spell check might still be able to reconstruct the original word.

Since these ideas and options are of statistical nature and need to be adapted to every specific application, they are not implemented or evaluated in the scope of this thesis. For example Tesseract and Finereader are equipped with a broad range of dictionaries and are even able to detect the underlying language. All in all, statistics house a broad range of possibilities and can increase recognition performance, but once implemented decreases the direct adaptability to other systems or languages.

7.2 Adaptability

In the previous section we've talked about the adaptability of the proposed algorithm. Further we have seen that the reCaptcha system changed and now features a new, different challenge of number recognition. As mentioned in the introduction also other competing captcha systems are available. We will try to adapt the proposed algorithm to those and see if it holds the potential in recognizing those.

In general the algorithm is implemented very flexibly. The challenge-extraction, desloping and deslanting process are three separate algorithms, which can also be left out. Also, the classification engine is solely based on instances of different characters. No retraining is needed once these are exchanged. In fact, the current implementation just loads all available images from a pattern-folder and uses those as the reference characters. Adapting the algorithm to e.g. numbers or alpha-numeric captchas should be fairly easy. Furthermore, the adaptation to classic, unmerged captchas should be possible by only allowing cutting hypotheses at blank spaces, instead of minima.

We will first start by testing the algorithm against the new reCaptchas consisting only of **numbers**. This shall not be an exhaustive evaluation, but rather show its adaptability and potential. Figure 7.3 shows some more instances of this type.

FIGURE 7.3: New version of the reCaptcha system as of fall 2013.

One can see that the numbers are again written on a curve and are slightly rotated or slanted. We could use the desloping- and deslanting-algorithms to correct this. Nevertheless, approximating the baseline of numbers is not as "easy" as of characters. Considering the consecutive "44" in Figure 7.3, the baseline is not anymore the lowest pixel. Also the deslanting-algorithm would run into heavy problems, since it is based on vertical stems inside the characters. Sadly those are not as frequent as they are in letters; most of the numbers only consist of round, circular objects.

Fortunately, both operations are not needed, as they were used to enable straight cuts at cutting-hypotheses. As one can see, the numbers are already separable via straight cuts.

As reference rendered versions of all numbers from the Font “Lucidia Sans”, rotated by up to ±5° are given. Again, numbers are per se not invariant. This is well known from a 6 and 9. Moreover, a 4 also looks almost exactly like a 9 if one smoothes its outline. Also a 2 and 5 are the same when flipped. Interestingly, the system seems to know about this; it seems like those ambigue 6,9,4,5,2 are way more present as easily detectable and unique 8 (the only number with two holes) and 3.

To overcome this ambiguity we first used the hole locations. We can again utilize this for 6 and 9, to detect whether the hole is present in the lower or upper part. Sadly this fails with the number 4. The second used method was to sent in probes, to detect the opening-direction of counters. This could be used for discriminating 2 and 5. Sadly both are opened to both the same sides. We will therefore quickly adapt the algorithm further. A 9 and 4 can be discriminated by their counters again, a 9 is opened west and a 4 to the east. To further distinguish between 2 and 5, we will also evaluate the previously unused location of the probe that dunked in the most. The counter of a 5 can be open to the east or west, but an east counter must be in the upper half, a west in the lower half—vice versa for a 2.

Figure 7.4 shows the first results of the algorithm with the mentioned approaches. We conclude that it has the potential to be adaptable to this system. The bottleneck appears to be again the segmentation algorithm; once the numbers are correctly cut, they are also classified correctly. Figure 7.5 shows a number-captcha where some of those problems are evident.

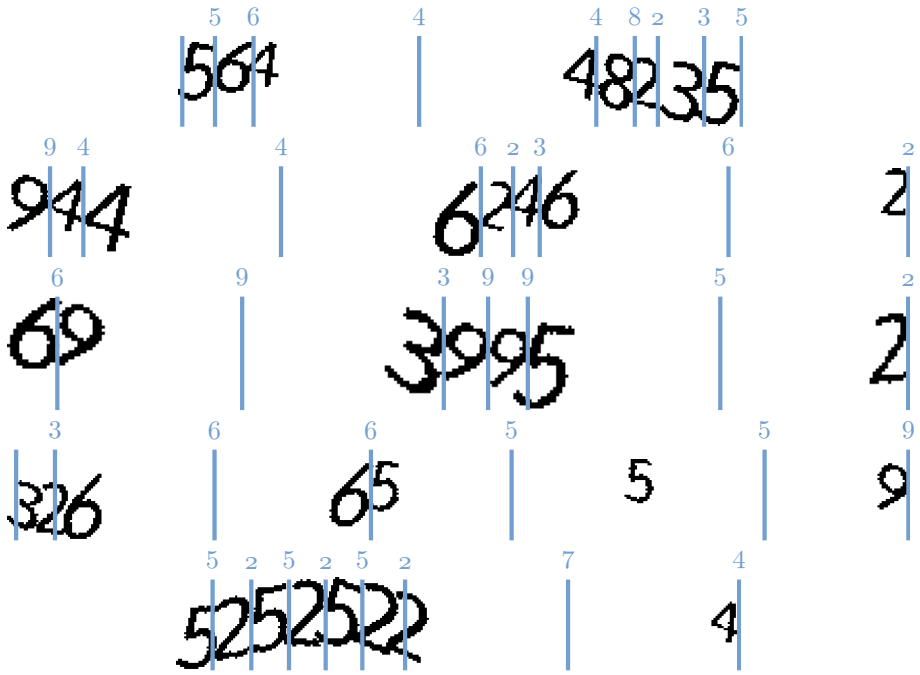


FIGURE 7.4: Deciphered number-captchas of the new reCaptcha system.

FIGURE 7.5: Problems in solving the number-captchas. 2 and 7 are very similar. The wrong classified 4 is very small and thus the hole-location is detected wrong.

For showing the adaptability to other systems we will try to decipher the Microsoft-captcha. As it is widely used among various Microsoft web-applications, it should represent a rather interesting

asset.

The Microsoft-captcha does not feature connected characters. As we therefore do not need to oversegment the image, all preprocessing steps can be ignored and it will only be cut between the stretched areas. As it is an alpha-numeric captcha, the number-instances are added to the previous reference-set and the previously mentioned adaptions for distinguishing numbers are in place. The Figure 7.6 shows how well the algorithm adapts to this type.

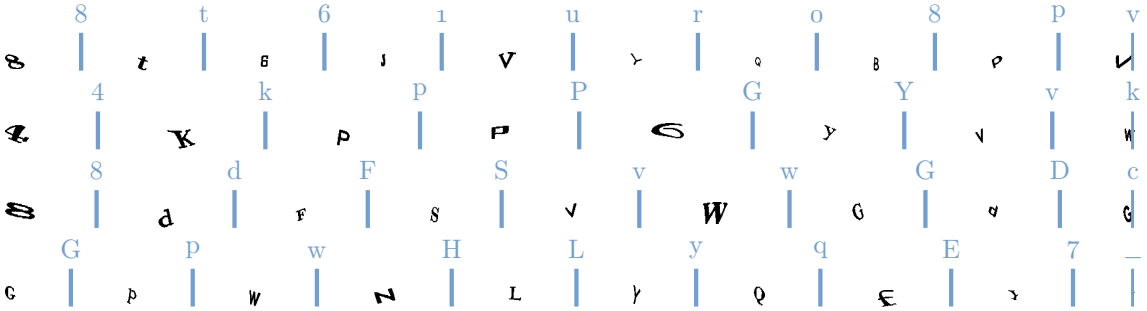


FIGURE 7.6: Deciphered Microsoft captchas.

7.3 Conclusion

Prior to coming to a final conclusion, we will discuss the limits of the developed algorithm.

Although we showed that the proposed algorithm is highly adaptable, some limits remain which include absolute thresholds. Even though we tried to refrain from using absolute thresholds and use relative thresholds, the algorithm is fixed to a minimum character height, maximum character width and a maximum hole width in pixels. We could circumvent these absolute values by requiring a constant text height. This could be implemented by considering the baseline and the topline as the first and last pixel per column. The average distance between both could then deal as a measurement to scale the segment to a constant height.

Another limit is that the algorithm always tries to deslope and deslant the segment, although it might not be needed. For a completely straight text segment, the desloping algorithm would still try to deslope it and as a result of this, some characters that stick out of the baseline ('p' or 'q') are deformed. Further, the deslanting process might overcorrect the already straight segment. Therefore, it should first be evaluated whether an preprocessing is really needed.

Furthermore, the algorithm strictly depends on the outline of characters. Once the outline breaks, or the characters are not classifiable by their outline, it will fail.

Nevertheless, we developed a highly adaptable algorithm that is capable of deciphering even very hard distortions.

We have shown that one of the last unbroken captcha systems is attackable by more-or-less simple techniques. With a rate of 30% we have beaten current profound systems and supplied an attack that renders this version of the captcha useless, since it cannot distinguish between a human and a computer in almost one out of three cases.

Furthermore, we have shown that Fourier descriptors are very useful in classifying heavily distorted characters. We have chosen them because of their heavy invariances that were needed to even detect such degraded characters. On the other hand we have encountered and shown that their heavy invariances are not necessarily good when applied to characters. We have further shown that a necessary necessity matching is able to circumvent this problem and utilize Fourier descriptors for character classification. We have supplied a classification engine that outperforms current OCR systems, especially on heavily distorted single characters.

Moreover, the developed algorithm only needs pre-rendered reference characters to classify individual characters. With this it is even adaptable to other character sets like the Cyrillic alphabet or even Chinese or Japanese character sets. It should be adaptable to any character set, as long as their characters are classifiable by their outline and the presence of holes. I.e. the shape of a hole is not important in deciding the characters. It should be an interesting research topic if the proposed engine is adaptable to Asian characters.

Finally, we conclude that we have achieved the goal of developing a robust recognition system. We also achieved the goal of breaking the reCaptcha system, without tailoring the system to it. We hope that this research will help in digitalizing old, degraded books. A pre-print of this thesis has been made available to Google, the reCaptcha-team, the Tesseract team and Abbyy.

As more and more captcha systems get broken, it is questionable whether a (text-only based) captcha system remains save and usable. As we have seen, reCaptcha changed its system that on the one hand disabled our approach, but on the other hand also extremely hardened the challenge for humans.

We will close with a quote from Ray Kurzweil from his book "Singularity is Near" [Kur05] about the prediction of artificial intelligence and the solving of the Turing test.

> "Since nonbiological intelligence will have passed the original Turing test years earlier (around 2029), should we allow a nonbiological human equivalent to be a judge?"
>
> – *Ray Kurzweil*

Appendix A

Fourier Descriptors

A.1 Mathematical Description and Generation of Fourier Descriptors

This shall give an exhaustive description of Fourier descriptors, their mathematical properties, derivations and invariances.

As no exhaustive derivation for all invariances and properties was found in the books, we will derive them from scratch. We require basic knowledge of well known properties of the (discrete) Fourier transformation and signal processing. We will follow roughly [Helo9], added with more derivations and a more convenient and usual nomenclature.

Nomenclature We define a function $a(n)$ in the discrete time domain as a lowercase function with the discrete time index n. The corresponding discrete Fourier transformation (DFT) is denoted by its uppercase function $A(k)$ with the discrete frequency index k.

The DFT is then defined as

$$a(n) \circ\!\!-\!\!\bullet A(k) \tag{A.1}$$

$$\mathfrak{F}\{a(n)\} = A(k) = \sum_{n=0}^{N-1} a(n) \exp\left(-\jmath 2\pi \frac{n}{N} k\right) \tag{A.2}$$

with $\jmath$ as the imaginary unit.

A.1.1 Generation of Fourier Descriptors

In order to transfer an outline of a shape into the frequency domain we first have to extract the boundary of the shape. For this very easy and profound methods exist in the books.

Prior to extracting the boundary a starting point needs to be selected in addition to an initial search direction. Extracting the boundary then leads to an ordered sequence of N real-valued coordinate points (x_N, y_N) with (x_0, y_0) denoting the starting point. We have chosen the starting point to always be the left most pixel in the vertical centre and the initial search direction to be north-west, i.e. clockwise.

This sequence of (real) coordinate points is then mapped into a complex contour-vector $a(n)$ by

$$a(n) = \begin{bmatrix} x_0 + \jmath\, y_0 \\ x_1 + \jmath\, y_1 \\ \vdots \\ x_{N-1} + \jmath\, y_{N-1} \end{bmatrix} \tag{A.3}$$

Onto $a(n)$ an one dimensional discrete Fourier transformation is applied, leading to N complex Fourier descriptors $A(k)$.

In order to low-pass the boundary, high-frequency components of $A(k)$ can then be discarded. Keep in mind that the high-frequency components are not directly the high-index coefficients, but lie in the middle. The DFT-vector starts with the zero-frequency DC offset, followed by the positive frequencies in the first half and contains the negative frequencies in the last half. Matlab provides a function `fftshift ()`, which centres the DC offset.

A.1.2 Invariances

For deriving the invariances, respectively the effects of certain operations on the Fourier descriptors we will utilize basic signal processing properties.

Translation Translating the complex contour $a(n)$ by a complex offset $z_x + \jmath z_y$ leads to change in the zero-frequency component (DC offset) in $A(k)$, i.e. a Dirac at frequency o is added.

$$a(n) + (z_x + \jmath z_y) \circ\!\!-\!\!\bullet\, A(k) + (z_x + \jmath z_y) \cdot \delta(0) \tag{A.4}$$

Scaling Scaling the complex contour $a(n)$ by a constant factor s leads to a global scaling factor in $A(k)$.

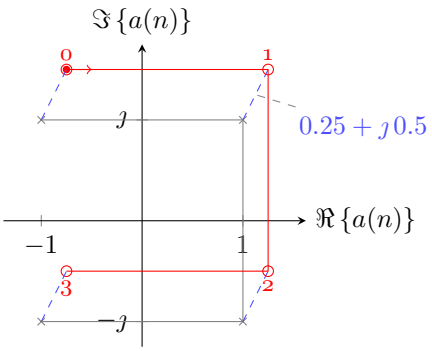
$$s \cdot a(n) \circ\!\!-\!\!\bullet\, s \cdot A(k) \tag{A.5}$$

Rotation Rotating the complex contour $a(n)$ by a constant angle ϕ, i.e. every complex-coordinate's phase is shifted by ϕ, leads to a constant phase shift in $A(k)$. This is the same effect as a scaling operation, since it is a multiplication by a constant.

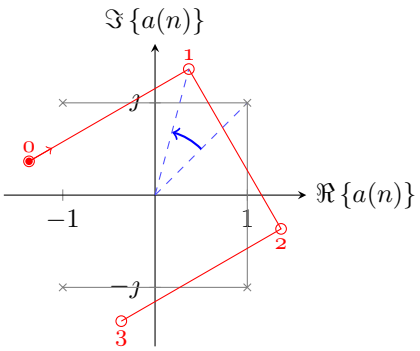
$$a(n) \cdot e^{\jmath\phi} \circ\!\!-\!\!\bullet\, A(k) \cdot e^{\jmath\phi} \tag{A.6}$$

Starting Point Shift Shifting the starting point $x_0 + \jmath y_0$ by m points, but—very important—keeping the search direction, leads to a phase change in the Fourier descriptors. For this derivation we utilize the shift theorem of the DFT.

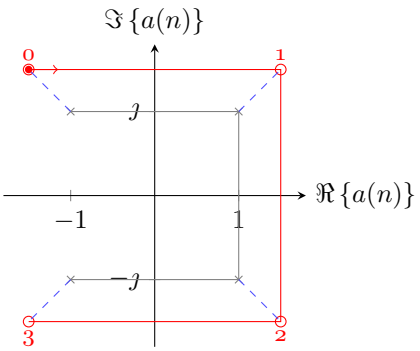
$$a(n - m) \circ\!\!-\!\!\bullet\, A(k) \cdot e^{\jmath 2\pi m \cdot k} \tag{A.7}$$



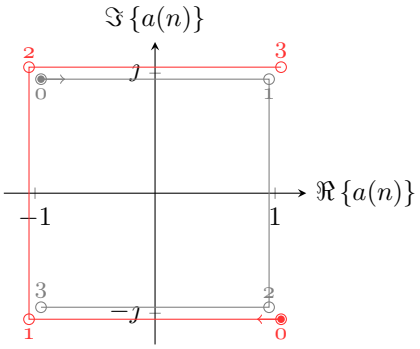
(A) Translation by $0.25 + \jmath 0.5$.



(B) Rotation by 30°.



(C) Scaling of $a(n)$ by 1.5.



(D) Shifting of the starting point by two indices. Note that the initial search direction stays the same (clockwise). For visualization both boundaries are slightly scaled.

FIGURE A.1: Different operations applied to the original complex contour $a(n)$ (grey) and the result $\tilde{a}(n)$ (red).

A.1.3 Effects of Mirroring and Further Derivations

While the previous effects of operations where very easy to show, deriving the impact of a mirroring operation is not that trivial.

In order to derive this, we'll first need to derive the effects of a contour-reversion and a complex-conjugation. While both are also elementary properties of the (discrete) Fourier-transformation, they are not that common.

Contour Reversal Reversing the order of a contour $a(n)$ leads to the contour $\tilde{a}(n) = a(-n)$. I.e. we change the initial search direction. Note that this is not a flipping of the vector like $a(N-1-n)$, but the first point—the starting point—stays fixed. Further, since the index periodically wraps around, $\tilde{a}(n) = a(-n) = a(N-n)$.

With this, the DFT derives as

$$\tilde{a}_{rev}(n) = a(-n) = a(N-n) \tag{A.8}$$

$$\mathfrak{F}\{a(N-n)\} = \sum_{n=0}^{N-1} a(N-n)\, e^{\left(-\jmath 2\pi \frac{n}{N} k\right)} , \tag{A.9}$$

substituting $n' = N - n$, the lower bound becomes $n = 0 \Rightarrow n' = N$ and the upper bound $n = N - 1 \Rightarrow n' = N - N + 1 = 1$

$$= \sum_{n'=1}^{N} a(n')\, e^{\left(-\jmath 2\pi \frac{N-n'}{N} k\right)} \tag{A.10}$$

$$= \sum_{n'=1}^{N} a(n')\, \underbrace{e^{\left(-\jmath 2\pi \frac{N}{N} k\right)}}_{=1 \forall k}\, e^{\left(-\jmath 2\pi \frac{-n'}{N} k\right)} \tag{A.11}$$

$$= \sum_{n'=1}^{N} a(n')\, e^{\left(-\jmath 2\pi \frac{n'}{N} (-k)\right)} \tag{A.12}$$

$$= \sum_{n'=0}^{N} a(n')\, e^{\left(-\jmath 2\pi \frac{n'}{N} (-k)\right)} - a(0) \tag{A.13}$$

$$= \sum_{n'=0}^{N-1} a(n')\, e^{\left(-\jmath 2\pi \frac{n'}{N} (-k)\right)} - a(0) + a(N) \underbrace{e^{\left(-\jmath 2\pi \frac{N}{N} (-k)\right)}}_{=1 \forall k} , \tag{A.14}$$

as $a(0) = a(N)$

$$= \sum_{n'=0}^{N-1} a(n')\, e^{\left(-\jmath 2\pi \frac{n'}{N} (-k)\right)} \tag{A.15}$$

$$= A(-k) . \tag{A.16}$$

In summary, a reversed complex-contour with fixed starting point leads to a reversion of the Fourier descriptors with the DC offset staying fixed.

$$a(-n) \circ\!\!-\!\!\bullet\, A(-k) \tag{A.17}$$

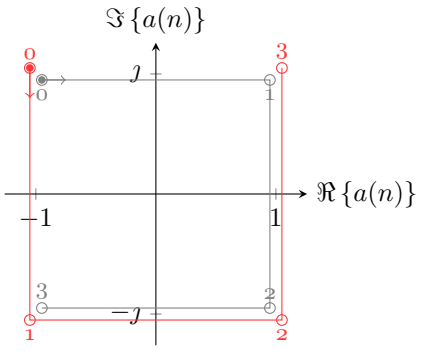


FIGURE A.2: Curve reversal $\tilde{a}(n)$ (red) of the contour $a(n)$ (grey). For visualization both boundaries are slightly scaled.

Complex Conjugation A complex conjugation on the contour $a(n)$, i.e. flipping all y_n, leads to a mirrored contour, in addition to a mirrored initial search direction.

$$\mathfrak{F}\{a(N)^*\} = \sum_{n=0}^{N-1} a(n)^* \; e^{-\jmath 2\pi \frac{n}{N} k} \tag{A.18}$$

$$= \sum_{n=0}^{N-1} \left(a(n) \; e^{\jmath 2\pi \frac{n}{N} k}\right)^* \tag{A.19}$$

$$= \left(\sum_{n=0}^{N-1} a(n) \; e^{\jmath 2\pi \frac{n}{N} k}\right)^* \tag{A.20}$$

$$= \left(\sum_{n=0}^{N-1} a(n) \; e^{-\jmath 2\pi \frac{n}{N} (-k)}\right)^* \tag{A.21}$$

$$= A(-k)^* \tag{A.22}$$

A complex conjugation of the contour leads to a complex conjugation of the Fourier descriptors in a reversed order.

$$a(n)^* \circ\!\!-\!\!\bullet \; A(-k)^* \tag{A.23}$$

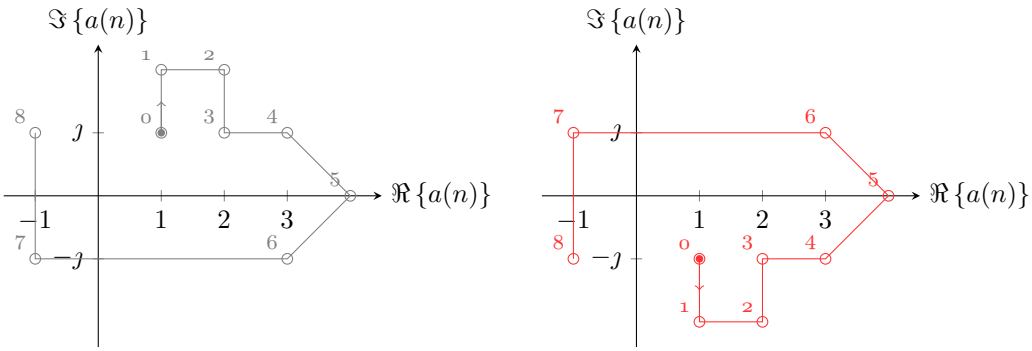


FIGURE A.3: Complex conjugation $\tilde{a}(n)$ (red, right) of the original contour $a(n)$ (grey, left). Notice that the curve orientation switches.

Vertical Mirroring A vertical mirrored (mirrored along the x-axis, or flipped up and down) contour $\tilde{a}(n)$ of $a(n)$ can be described as a complex conjugated and reversed contour. Both is needed, since we require that the starting point (its relative position in the contour) does not change and that the initial search direction stays the same. Therefore, $\tilde{a}(n) = a^*(-n)$ and with Equation A.23 and A.17 a mirroring again only has an influence in the phase:

$$\tilde{a}_{\updownarrow}(n) = a^*(-n) \circ\!\!-\!\!\bullet \; A(k)^* \tag{A.24}$$

Horizontal Mirroring Horizontal mirroring is derived analogue to vertical mirroring as $\tilde{a}_{\leftrightarrow}(n) = -\tilde{a}_{\updownarrow}(n) = -a^*(-n)$ and therefore is again just another phase change of π.

$$\tilde{a}_{\leftrightarrow}(n) = -a^*(-n) \circ\!\!-\!\!\bullet \; A(k)^* \cdot e^{-\jmath\pi} \tag{A.25}$$

A.2 Normalisation

As we have derived all influences, we are now able to normalise the Fourier descriptors in order to make them invariant.

Translation We have seen that a translation of the contour only influences the zero-frequency component. We therefore are able to centre the shape at the origin and remove any translation by setting the DC offset to zero.

$$A_{inv}(0) = 0 \tag{A.26}$$

Scaling As a scaling operation affects all components equally, we can normalise all descriptors by a common divisor, e.g. by the first frequency. Notice that a scaling operation affects the magnitude of the coefficient and not the phase. Therefore, we can achieve scaling invariance by normalising all descriptors by the magnitude of the first coefficient (not the DC offset).

$$A_{inv}(k) = \frac{A(k)}{|A(1)|} \tag{A.27}$$

Rotation, Starting Point and Mirroring We have seen that all three operations affect the phases of every coefficient in different degrees. We therefore could achieve complete invariance against these operations by removing the phase information. Unfortunately this has also the effect that the Fourier descriptors heavily generalize the shape and a lot of information is lost. Figure A.4 shows two different objects with the same magnitude spectrum.

As the phases are influenced differently by starting point shifts, rotations and mirroring, a comparison or normalisation is not trivial. As discussed in this thesis certain phase information can be used in order to allow a range of invariance, without loosing the complete phase information.

Furthermore, we have seen that mirroring affects the phases by a complex conjugation, which should be fairly easy to detect. Though this only holds true, while the starting point remains its relative position in the contour. Since that does not hold true, when always selecting an starting point at a fixed absolute position, it will lead to a starting point shift and therefore to a further change of the phases. Thus, detecting a mirroring by the complex conjugation is rendered almost impossible. It would however be very easy to detect if the starting point would be selected using a heuristic that would always select the same relative position. This is also discussed in Section 5.2.1.

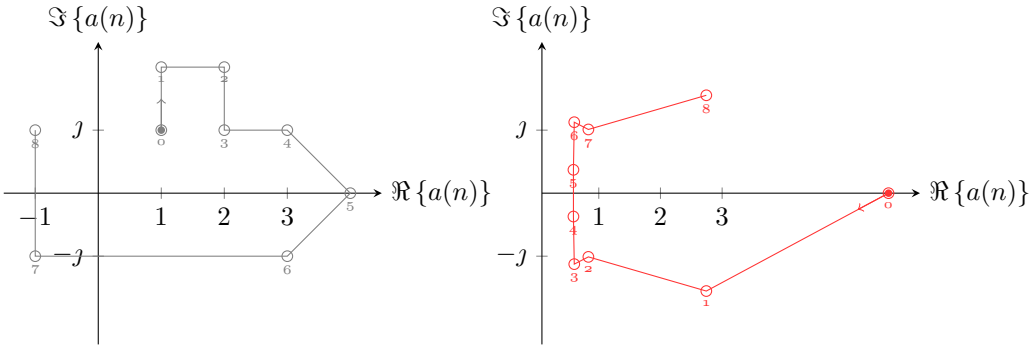


FIGURE A.4: Both objects share the same magnitude spectrum, i.e. the Fourier descriptors match if the phase information is discarded. Clearly both objects are nowhere near similar.

Appendix B

Confusion Matrices

Following all previously embedded confusion tables in detail. As lower- and uppercase characters are most times very similar they are consolidated.

All numbers are given in percent as, e.g. for the first table, "*98.46% of all 'a's are classified as 'a'. 1.54% of all 'a's are classified as 'd'.*" In other words, the numbers are *not* adjusted to the frequency of the individual letters in the set.

This becomes noticeable in the captcha sets, where sadly only four 'x' were acquirable. Hence the unusual distribution of 25%, 25% and 50%.

TABLE B.1: Confusion matrix of distorted times test set for proposed algorithm using 32 Fourier descriptors. (in %).

	a	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**i**	**j**	**k**	**l**	**m**	**n**	**o**	**p**	**q**	**r**	**s**	**t**	**u**	**v**	**w**	**x**	**y**	**z**
a	98.46	0	0	1.54	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
b	0	94.62	0	0.77	0	0	4.62	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
c	0	0	100.00	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
d	0	0	0	100.00	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
e	0	0	0	0	100.00	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
f	0	0	0	0	0	73.85	0	0	0	0	0	3.08	0	0	0	0	0	19.23	0	3.85	0	0	0	0	0	0
g	0	0	0	0	0	0	100.00	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
h	0	0	0	0	0	0	0	100.00	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
i	0	0	0	0	0	0	0	0	99.23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.77
j	0	0	0	0	0	0	0	0	0	98.46	0	0	0	0	0	0	0	0	0	0	0	0	0	1.54	0	0
k	0	0	0	0	0	0	0	0	0	0	96.15	0	0	0	0	0	0	0	0	0	0	0	0	3.85	0	0
l	0	0	0	0	0	0	0	0	23.08	0	0	76.92	0	0	0	0	0	0	0	0	0	0	0	0	0	0
m	0	0	0	0	0	0	0	0	0	0	0	0	97.69	0	0	0	0	1.54	0	0	0	0.77	0	0	0	0
n	0	0	0	0	0	0	0	0	0	0	0	0	0	100.00	0	0	0	0	0	0	0	0	0	0	0	0
o	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.00	0	0	0	0	0	0	0	0	0	0	0
p	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.00	0	0	0	0	0	0	0	0	0	0
q	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.00	0	0	0	0	0	0	0	0	0
r	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.00	0	0	0	0	0	0	0	0
s	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.00	0	0	0	0	0	0	0
t	0	0	0	0	0	0	0	0	0	0	0	0.77	0	0	0	0	0	16.15	0	83.08	0	0	0	0	0	0
u	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.00	0	0	0	0	0
v	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.00	0	0	0	0
w	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.00	0	0	0
x	0	0	0	0	0	0	0	0	0	0	6.92	0	0	0	0	0	0	0	0	0	0	0	0	93.08	0	0
y	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.00	0
z	0	0	0	0	0	0	0	0	0	0	3.08	0	0	0	0	0	0	0	0	0	0	0	0	0	0	96.92

TABLE B.2: Confusion matrix of distorted times test set for proposed algorithm using 16 Fourier descriptors. (in %).

	a	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**i**	**j**	**k**	**l**	**m**	**n**	**o**	**p**	**q**	**r**	**s**	**t**	**u**	**v**	**w**	**x**	**y**	**z**
a	98.46	0	0	1.54	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
b	0	94.62	0	0.77	0	0	4.62	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
c	0	0	100.00	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
d	0	0	0	100.00	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
e	0	0	0	0	100.00	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
f	0	0	0	0	0	70.00	0	0	0	0	0	2.31	0	0	0	0	0	18.46	0	9.23	0	0	0	0	0	0
g	0	0	0	0	0	0	100.00	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
h	0	0	0	0	0	0	0	100.00	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
i	0	0	0	0	0	0	0	0	99.23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.77
j	0	0	0	0	0	0	0	0	0	93.08	0	0	0	0	0	0	0	0	0	0	0	0	0	6.92	0	0
k	0	0	0	0	0	0	0	0	0	0	96.15	0	0	0	0	0	0	0	0	0	0	0	0	3.85	0	0
l	0	0	0	0	0	0	0	0	20.00	0	0	80.00	0	0	0	0	0	0	0	0	0	0	0	0	0	0
m	0	0	0	0	0	0	0	0	0	0	0	0	97.69	0	0	0	0	1.54	0	0	0	0.77	0	0	0	0
n	0	0	0	0	0	0	0	0	0	0	0	0	0	98.46	0	0	0	0	0	0	0	0	0	0	0	1.54
o	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.00	0	0	0	0	0	0	0	0	0	0	0
p	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	99.23	0.77	0	0	0	0	0	0	0	0	0
q	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.00	0	0	0	0	0	0	0	0	0
r	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.00	0	0	0	0	0	0	0	0
s	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.00	0	0	0	0	0	0	0
t	0	0	0	0	0	0	0	0	0	0	0	0.77	0	0	0	0	0	21.54	0	77.69	0	0	0	0	0	0
u	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.00	0	0	0	0	0
v	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.00	0	0	0	0
w	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.00	0	0	0
x	0	0	0	0	0	0	0	0	0	0	6.92	0	0	0	0	0	0	0	0	0	0	0	0	93.08	0	0
y	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.00	0
z	0	0	0	0	0	0	0	0	0	0	3.08	0	0	0	0	0	0	0	0	0	0	0	0	0	0	96.92

TABLE B.3: Confusion matrix of distorted times test set for Tesseract. (in %).

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	?
a	87.69	0	0	0	0	0	0	5.38	4.62	0	0	0	0	0	2.31	0	0	0	0	0	0	0	0	0	0	0	0
b	0	47.69	0	0	0	0	0	0	6.92	0	3.85	0.77	0	2.31	0	0	0	0	0.77	0	0	0	0	28.46	1.54	0.77	6.92
c	0	0	98.46	0	0	0	0	0	0	0	0	0	0	0	0	0	1.54	0	0	0	0	0	0	0	0	0	0
d	0.77	1.54	0	89.23	0	0	0	0	8.46	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
e	0	2.31	3.08	0	63.08	0	1.54	0	29.23	0	0	0	0	0	0	0	0.77	0	0	0	0	0	0	0	0	0	0
f	0	0	0	0	0	95.39	0	0	0.77	0	3.85	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
g	0	0	0	0	0	0	53.08	0	0	0	0.77	0	0	3.08	0	0	0	0	0	0	0	1.54	0.77	11.54	6.92	0.77	21.54
h	0	1.54	0	0	0.77	0	0	96.92	0	0	0	0	0	0	0	0	0.77	0	0	0	0	0	0	0	0	0	0
i	0	0	0	0	0	0	0	0	48.46	0	0	0	0	5.38	0	0	0	0	0	0	0	0	0	33.85	0	0	12.31
j	0	0	0	0	0	0	0	0	1.54	47.69	3.08	0	0	10.77	0	0	0	0.77	0.77	0	0	0	0	20.77	0	0	14.62
k	0	0	0	0	0	0	0	0	0	0	100.00	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
l	0	0	0	0	0	0	0	0	0	0	0	95.39	0	0	0	0	0	0	0	0	0	0	0	4.62	0	0	0
m	0	0	0	0	0	0	0	0	1.54	0	0	0	94.62	0	0	0	0	0.77	0	0	0	0	1.54	0	0	0	1.54
n	0	2.31	0	2.31	0	0	0	33.08	0	0	0	0	0	53.85	0	0	3.08	0	0	0	0	0	0	0	0	0	5.38
o	0	0	0	0	0	0	0	0	10.77	0	0	0	0	0	86.15	0	3.08	0	0	0	0	0	0	0	0	0	0
p	0	0	0	0.77	0	0	0	0	38.46	0	0	0.77	0	2.31	3.85	52.31	0.77	0	0	0	0	0	0	0	0	0	0.77
q	3.08	0	5.38	0	0	0	0	0	14.62	0	0	0.77	0	0	2.31	0	70.77	0	2.31	0.77	0	0	0	0	0	0	0
r	0	0	0	0	0	23.08	0	0	11.54	0	0	0	0	0	0	7.69	0	36.92	0	0.77	1.54	0	0	0	18.46	0	0
s	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.00	0	0	0	0	0	0	0	0
t	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	99.23	0	0	0	0.77	0	0	0
u	0	0	0	0	0	0	0	6.92	0	0	0	0	0	0	0	0	3.85	0	0	0	87.69	0	0	0	0	0	1.54
v	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.00	0	0	0	0	0
w	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.00	0	0	0	0
x	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.00	0	0	0
y	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.77	0	0	99.23	0	0
z	0	0	0	0	0	0	0	0	6.15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	93.85	0

TABLE B.4: Confusion matrix of captcha test set for Finereader 11. (in %).

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	?
a	19.44	0	0	0	0.93	0	0	0	0	0	0	0.93	0	0	0	0	0	0	0.93	1.85	0	0	0	0	0	0.93	75.00
b	2.08	25.00	0	0	0	0	0	6.25	4.17	2.08	0	0	0	0	0	0	2.08	2.08	0	4.17	0	0	0	0	0	0	52.08
c	0	0.95	49.52	0.95	0.95	0	0	0	2.86	0.95	0	0	0	0	0.95	0	0	0.95	0	0	0	0	0	0	0	0	41.91
d	3.06	0	1.02	26.53	0	2.04	0	0	2.04	1.02	0	1.02	0	0	0	0	2.04	1.02	0	1.02	0	0	0	0	0	0	59.18
e	0	1.24	0.41	0	19.92	0.83	0	0	1.66	0.41	0	0	0	0	0	0	0.41	0.41	0.41	0.41	0	0	0	0	0	0	73.86
f	0	0	0	0	2.33	18.61	0	0	9.30	0	0	0	0	0	0	0	0	2.33	0	6.98	0	0	0	0	0	0	60.47
g	0	0	0	0	0	0	10.00	0	0	3.33	0	0	0	0	0	0	0	0	3.33	0	0	0	0	0	0	0	83.33
h	0	0	1.27	0	0	1.27	0	43.04	1.27	0	0	1.27	0	1.27	0	0	0	0	0	0	0	1.27	0	0	1.27	0	48.10
i	1.70	0	0	0	0	2.55	0	0.43	17.02	0.85	0	22.98	1.70	0.43	0	0	0	1.28	1.70	2.98	0	0.85	2.13	1.28	0	0	42.13
j	14.29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14.29	0	0	0	0	0	0	0	71.43
k	0	0	5.00	0	0	0	0	0	0	0	10.00	5.00	0	0	0	0	0	0	0	10.00	5.00	0	0	0	5.00	0	60.00
l	0	0	0	0	0	0.78	0	1.55	10.85	0	0.78	20.93	0	0.78	0	0	0	0.78	0.78	3.88	0	0	0	0.78	0.78	0.78	56.59
m	2.20	0	3.30	0	0	2.20	0	0	5.49	1.10	2.20	2.20	14.29	5.49	2.20	0	0	10.99	0	6.59	0	2.20	0	2.20	1.10	0	36.26
n	0.63	0	0	1.90	0	2.53	0	0.63	0	0.63	0	4.43	0	24.68	0	0	0	1.27	0.63	6.33	0	0	0.63	1.27	1.27	0	53.17
o	4.27	1.22	0.61	7.32	0.61	0.61	0	0	1.83	0	0	1.83	0	1.22	30.49	0.61	2.44	0.61	1.83	1.22	0	0	0	0	0	0.61	42.68
p	1.75	1.75	0	0	0	1.75	0	0	5.26	0	0	0	0	1.75	1.75	29.83	0	0	0	0	1.75	1.75	0	0	0	0	52.63
q	20.00	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20.00	0	0	0	0	0	0	0	0	0	60.00
r	0	0	0.64	0	1.27	0.64	0	0	1.91	0.64	0	1.27	0	1.27	0	0.64	0	33.12	0	8.92	0	0.64	0	1.27	0	0	47.77
s	0.64	0	0	0	0	0	0	0	0.64	0	0	0	0	0	0	0	0	0	9.62	0	0	0	0	0	0	0	89.10
t	1.04	0	0.52	0	1.04	0	0	0.52	3.11	1.04	0	2.59	0.52	0.52	0	0	0	0	0.52	34.20	0	0	0	0	0	0	54.40
u	2.20	0	0	0	0	1.10	0	1.10	2.20	0	1.10	2.20	0	1.10	1.10	0	0	0	1.10	2.20	21.98	3.30	0	1.10	0	0	58.24
v	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4.76	52.38	0	0	0	0	42.86
w	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/
x	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.0
y	0	0	0	0	3.03	0	0	0	0	0	0	0	0	0	0	0	0	0	3.03	0	0	6.06	0	0	57.58	0	30.30
z	0	0	0	0	0	0	0	0	9.09	0	0	0	0	0	0	9.09	0	0	0	0	0	0	0	0	0	27.27	54.55

TABLE B.5: Confusion matrix of captcha test set for Tesseract. (in %).

	a	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**i**	**j**	**k**	**l**	**m**	**n**	**o**	**p**	**q**	**r**	**s**	**t**	**u**	**v**	**w**	**x**	**y**	**z**	**?**
a	21.91	7.62	0.95	0	17.14	3.81	0.95	2.86	2.86	1.90	1.90	0	0	10.48	0	0	5.71	7.62	2.86	0	0.95	0	0	2.86	0	0.95	6.67
b	0	38.64	4.55	0	2.27	2.27	0	6.82	4.55	0	4.55	0	0	9.09	2.27	0	0	0	0	0	0	2.27	0	13.64	2.27	0	6.82
c	0	2.00	49.00	1.00	2.00	1.00	9.00	0	0	0	0	0	0	2.00	0	0	13.00	1.00	0	0	2.00	0	0	13.00	0	0	5.00
d	2.13	0	3.19	42.55	1.06	1.06	2.13	0	11.70	0	0	0	0	4.26	3.19	0	6.38	0	4.26	0	2.13	1.06	0	7.45	0	1.06	6.38
e	0.43	11.11	0.85	0	35.90	0.43	8.97	0	0	0	3.85	0	0	6.41	0.43	0.43	7.69	0.85	1.28	0	0	0	0.43	9.83	0	0.43	10.68
f	0	0	0	0	6.98	30.23	0	2.33	23.26	0	6.98	0	0	0	0	0	2.33	4.65	4.65	4.65	0	0	0	0	0	0	13.95
g	0	0	0	0	0	0	17.24	0	0	0	0	0	0	6.90	0	0	3.45	0	0	0	0	0	0	24.14	6.90	3.45	37.93
h	0	2.63	0	0	3.95	2.63	0	73.68	0	0	6.58	0	0	1.32	0	0	0	0	0	0	0	1.32	0	2.63	0	0	5.26
i	0	0	0	0	3.74	0.93	0	0	18.69	0	0	0	0	6.54	0	0	0	0	1.40	0	0	0	0	31.31	0	0	37.38
j	0	0	0	0	0	0	0	0	0	28.57	0	0	0	14.29	0	0	0	0	0	0	0	0	0	14.29	0	0	42.86
k	0	0	0	0	0	0	0	11.77	0	0	52.94	0	0	5.88	0	0	0	17.65	0	0	5.88	0	0	5.88	0	0	0
l	0.81	0	0	0	3.23	1.61	0.81	0	7.26	0	2.42	58.07	0	3.23	0	0	2.42	0	1.61	0	0	0.81	0	8.06	0	0.81	8.87
m	0	0	0	0	0	0	0	0	0	0	0	0	55.39	15.39	0	0	0	0	0	3.08	0	0	10.77	4.62	0	0	10.77
n	0.70	4.23	0	12.68	1.41	1.41	1.41	30.28	0	0	2.82	0.70	0	7.04	1.41	0	5.63	4.93	0	0	4.93	0.70	0.70	7.04	0	0	11.97
o	0	0.65	1.95	9.74	0	0.65	1.95	0.65	11.04	0	1.95	0	0.65	7.79	25.97	0	10.39	0	0.65	1.95	7.14	0	0.65	8.44	0	0	7.79
p	0	1.85	1.85	11.11	1.85	5.56	1.85	1.85	9.26	0	7.41	0	1.85	3.70	3.70	18.52	14.82	0	3.70	1.85	0	0	0	3.70	0	0	5.56
q	0	0	20.00	0	0	0	0	0	0	0	0	0	0	40.00	0	0	40.00	0	0	0	0	0	0	0	0	0	0
r	0	1.30	3.90	0.65	4.55	16.88	0	0.65	25.97	0	1.30	0	0	1.30	0	7.79	1.95	3.90	0.65	14.94	0	0	0	4.55	3.90	0	5.84
s	0	1.29	0	0	3.23	1.29	0.65	0	0	0	0.65	0	0	9.03	0	0	0.65	1.29	63.23	0	0	0	0	9.68	0	0	9.03
t	0	0	2.17	0	4.35	1.63	0	0	13.04	0.54	5.43	1.09	0	0.54	0	0	1.09	0	0.54	55.44	0	0	0.54	7.07	0	0.54	5.98
u	0	1.16	0	10.47	1.16	0	0	10.47	0	2.33	1.16	0	2.33	1.16	1.16	0	4.65	0	0	0	52.33	0	0	5.81	0	0	5.81
v	0	0	0	4.76	0	0	0	0	0	0	4.76	0	0	4.76	0	4.76	9.52	0	0	0	0	61.91	0	4.76	0	0	4.76
w	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/
x	0	0	0	0	0	0	0	0	25.00	0	0	0	0	25.00	0	0	0	0	0	0	0	0	0	50.00	0	0	0
y	0	0	0	0	0	3.13	0	0	3.13	0	9.38	0	0	9.38	0	3.13	3.13	3.13	6.25	6.25	0	6.25	0	0	34.38	0	12.50
z	0	0	0	0	0	0	0	0	0	0	0	0	10.00	10.00	0	0	10.00	0	0	0	0	0	0	0	0	50.00	20.00

TABLE B.6: Confusion matrix of captcha test set for proposed algorithm utilizing 32 Fourier descriptors. (in %).

	a	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**i**	**j**	**k**	**l**	**m**	**n**	**o**	**p**	**q**	**r**	**s**	**t**	**u**	**v**	**w**	**x**	**y**	**z**	**?**
a	87.62	0	0	4.76	0	0	0	0	0	0	0	0	0	0	0	0	0.95	0.95	5.71	0	0	0	0	0	0	0	0
b	0	81.82	0	13.64	0	0	4.55	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
c	0	0	90.00	0	0	0	6.00	0	0	1.00	0	1.00	0	0	0	0	0	1.00	0	1.00	0	0	0	0	0	0	0
d	1.06	2.13	0	93.62	0	0	0	0	0	0	0	0	0	0	0	0	2.13	1.06	0	0	0	0	0	0	0	0	0
e	0.85	0.43	0.43	0	90.17	0	0	0	0	0	0	0	0	0	0	1.71	0.43	1.71	2.99	0.43	0	0	0.43	0	0	0	0.43
f	0	0	0	0	0	32.56	0	0	0	4.65	0	18.61	0	0	0	0	0	0	0	34.88	2.33	0	4.65	0	2.33	0	0
g	0	0	0	0	3.45	0	96.55	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
h	0	0	0	0	0	0	0	77.63	0	0	3.95	0	0	14.47	0	0	0	2.63	0	0	1.32	0	0	0	0	0	0
i	0	0	0	0	0	2.80	0	0	76.64	7.48	0	8.88	0	0	0	0	0	2.34	0	0.93	0	0	0	0	0.93	0	0
j	0	0	0	0	0	0	0	0	14.29	85.71	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
k	0	0	0	0	0	0	0	5.88	0	0	70.59	0	0	0	0	0	0	0	0	0	0	0	5.88	17.65	0	0	0
l	0	0	0	0	0	6.45	0	0	0.81	4.84	0	80.65	0	0	0	0	0	1.61	0	4.84	0	0.81	0	0	0	0	0
m	0	0	0	0	0	3.08	0	0	0	0	0	0	95.39	0	0	0	0	0	0	0	0	0	1.54	0	0	0	0
n	0	0	0.70	0	0	0.70	0	3.52	0	0.70	1.41	0.70	0.70	91.55	0	0	0	0	0	0	0	0	0	0	0	0	0
o	0	2.60	0	27.92	0	0	0	0	0	0	0	0	0	0	64.29	0	3.25	0.65	0	0	0	0	0.65	0	0	0	0.65
p	0	3.70	0	11.11	0	0	0	0	0	0	0	0	0	0	0	81.48	0	3.70	0	0	0	0	0	0	0	0	0
q	0	0	0	20.00	0	0	0	0	0	0	0	0	0	0	0	0	80.00	0	0	0	0	0	0	0	0	0	0
r	0	0	1.95	0	0	3.25	0	0	2.60	0	1.30	11.69	0	0.65	0	0	0	59.74	1.30	16.23	0	0	0	0	1.30	0	0
s	0	0	0.65	0	0	3.23	0.65	0	0.65	1.29	4.52	0	0.65	0	0	0	0	0	83.87	0.65	0	0	1.29	0	0.65	1.29	0.65
t	0	0	0.54	0	0	4.35	0	0	2.17	1.09	0.54	9.24	0	0	0	0	0	7.07	0	72.28	0	0	0	0	1.63	1.09	0
u	0	0	1.16	0	0	0	0	0	0	0	0	0	0	1.16	0	0	0	0	0	0	94.19	3.49	0	0	0	0	0
v	0	0	0	0	0	0	0	0	0	0	0	14.29	0	0	0	0	0	9.52	0	0	0	71.43	0	0	4.76	0	0
w	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/
x	0	0	0	0	0	25.00	0	0	0	0	0	0	0	0	0	0	0	25.00	0	0	0	0	0	50.00	0	0	0
y	0	0	0	0	0	0	0	0	0	0	3.13	0	0	0	0	0	0	3.13	0	0	0	0	0	0	93.75	0	0
z	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20.00	10.00	0	0	0	0	0	70.00	0

TABLE B.7: Confusion matrix of captcha test set for proposed algorithm utilizing 16 Fourier descriptors. (in %).

	a	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**i**	**j**	**k**	**l**	**m**	**n**	**o**	**p**	**q**	**r**	**s**	**t**	**u**	**v**	**w**	**x**	**y**	**z**	**?**
a	89.52	0	0	3.81	0	0	0	0	0	0	0	0	0	0	0	0	0	0.95	5.71	0	0	0	0	0	0	0	0
b	0	81.82	0	13.64	0	0	4.55	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
c	0	0	90.00	0	0	0	7.00	0	0	1.00	0	0	0	0	0	0	0	1.00	0	1.00	0	0	0	0	0	0	0
d	1.06	2.13	0	93.62	0	0	0	0	0	0	0	0	0	0	0	0	2.13	1.06	0	0	0	0	0	0	0	0	0
e	1.28	0.43	0.85	0	89.32	0	0	0	0	0	0	0.43	0	0	0	1.71	0.43	1.28	3.42	0.43	0	0	0	0	0	0	0.43
f	0	0	0	0	0	34.88	0	0	0	4.65	0	16.28	0	0	0	0	0	0	0	34.88	2.33	0	4.65	0	2.33	0	0
g	0	0	0	0	3.45	0	96.55	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
h	0	0	0	0	0	0	0	78.95	0	0	2.63	0	0	14.47	0	0	0	1.32	0	0	1.32	0	0	0	0	1.32	0
i	0	0	0	0	0	2.80	0	0	76.17	7.94	0	8.88	0	0	0	0	0	2.34	0	0.93	0	0	0	0	0.93	0	0
j	0	0	0	0	0	0	0	0	14.29	85.71	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
k	0	0	0	0	0	0	0	5.88	0	0	70.59	0	0	0	0	0	0	0	0	0	0	0	5.88	17.65	0	0	0
l	0	0	0	0	0	8.87	0	0	0.81	4.84	0	79.84	0	0	0	0	0	1.61	0	3.23	0	0.81	0	0	0	0	0
m	0	0	0	0	0	1.54	0	0	0	0	0	0	96.92	0	0	0	0	0	0	0	0	0	1.54	0	0	0	0
n	0	0	0.70	0	0	0.70	0	4.23	0	0	1.41	0.70	0.70	91.55	0	0	0	0	0	0	0	0	0	0	0	0	0
o	0	2.60	0	29.22	0	0	0	0	0	0	0	0	0	0	62.99	0	3.25	0.65	0	0	0	0	0.65	0	0	0	0.65
p	0	3.70	0	14.82	0	0	0	0	0	0	0	0	0	0	0	79.63	0	1.85	0	0	0	0	0	0	0	0	0
q	0	0	0	20.00	0	0	0	0	0	0	0	0	0	0	0	0	80.00	0	0	0	0	0	0	0	0	0	0
r	0	0	1.95	0	0	1.95	0	0	2.60	0	1.30	11.04	0	0.65	0	0	0	58.44	1.30	19.48	0	0	0	0	1.30	0	0
s	0	0	0.65	0	0	3.23	0.65	0	0.65	2.58	5.16	0	0	0	0	0	0	0	83.23	0.65	0	0	1.29	0	1.29	0	0.65
t	0	0	0.54	0	0	5.43	0	0	2.17	1.09	0.54	9.24	0	0	0	0	0	7.61	0	70.65	0	0	0	0	1.63	1.09	0
u	0	0	1.16	0	0	0	0	0	0	0	0	0	0	1.16	0	0	0	0	0	0	91.86	5.81	0	0	0	0	0
v	0	0	0	0	0	0	0	0	0	0	0	19.05	0	0	0	0	0	4.76	0	0	0	71.43	0	0	4.76	0	0
w	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/
x	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	50.00	0	0	0	0	0	50.00	0	0	0
y	0	0	0	0	0	0	0	0	0	0	3.13	0	0	0	0	0	0	3.13	0	0	0	0	0	0	93.75	0	0
z	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20.00	10.00	0	0	0	0	0	70.00	0

Bibliography

[ABHL03] Luis Von Ahn, Manuel Blum, Nicholas J Hopper, and John Langford: *CAPTCHA: Using hard AI problems for security.* Advances in Cryptology–EUROCRYPT 2003, pages 294–311, 2003.

[AMM+08] Luis Von Ahn, Benjamin Maurer, Colin McMillen, David Abraham, Manuel Blum, and Luis von Ahn: *reCAPTCHA: human-based character recognition via Web security measures.* Science (New York, N.Y.), 321(September):1465–1468, September 2008, ISSN 1095-9203.

[BBFM11] Paul Baecher, Niklas Büscher, Marc Fischlin, and Benjamin Milde: *Breaking reCAPTCHA : A Holistic Approach via Shape Recognition.* In *Future Challenges in Security and Privacy for Academia and Industry*, pages 56–67. Springer, 2011.

[BMM11] Elie Bursztein, Matthieu Martin, and John C. Mitchell: *Text-based CAPTCHA strengths and weaknesses.* ACM Computer and Communication security 2011, CSS 2011, 2011.

[Bur12] Elie Bursztein: *The art of breaking and designing captchas.* In *RSA Conference*, 2012.

[CLSC05a] Kumar Chellapilla, Kevin Larson, Patrice Simard, and Mary Czerwinski: *Computers beat Humans at Single Character Recognition in Reading based Human Interaction Proofs (HIPs).* CEAS, 98053, 2005.

[CLSC05b] Kumar Chellapilla, Kevin Larson, Patrice Simard, and Mary Czerwinski: *Designing Human Friendly Human Interaction Proofs (HIPs).* SIGCHI Conference on Human Factors in Computing Systems, pages 711–720, 2005.

[Cod] Google Code: *Issue 748 - tesseract-ocr - Missing the x_ wconf value when using hOCR in the last version.* `https://code.google.com/p/tesseract-ocr/issues/detail?id=748`, visited on 25.11.2013.

[Dro03] Michael Droettboom: *Correcting broken characters in the recognition of historical printed documents.* Joint Conference on Digital Libraries, 2003.

[FHC05] KC Fan, CH Huang, and TC Chuang: *Italic detection and rectification.* Journal of Information Science and Engineering, 23(91):403–419, 2005.

[Gut] Project Gutenberg: *Free ebooks - project gutenberg.* `http://www.gutenberg.org/`, visited on 25.11.2013.

[GWF12] Haichang Gao, Wei Wang, and Ye Fan: *Divide and Conquer: An Efficient Attack on Yahoo! CAPTCHA.* 2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications, pages 9–16, June 2012.

[GWF+13] Haichang Gao, Wei Wang, Ye Fan, Jiao Qi, and Xiyang Liu: *The robustness of "Connecting Characters Together" CAPTCHAs.* Unpublished, 2013. `http://journal.iis.sinica.edu.tw/paper/1/12008-2.pdf?cd=2CA6D3FBF52CB99DE`.

[Hel09] Olaf Hellwich: *Fourier Descriptors.* Lecture, Berlin University of Technology, 2009.

[HLBO10] Shih Yu Huang, Yeuan Kuen Lee, Graeme Bell, and Zhan he Ou: *An efficient segmentation algorithm for CAPTCHAs with line cluttering and character warping.* Multimedia Tools and Applications, pages 1–25, 2010.

[Hoc] Sam Hocevar: *PWNtcha – Caca Labs.* `http://caca.zoy.org/wiki/PWNtcha`, visited on 25.11.2013.

[Kal] Gursev Singh Kalra: *Attacking Visual CAPTCHAs with TesserCap.* Foundstone Whitepaper.

[KFK00] Ergina Kavallieratou, Nikos Fakotakis, and G Kokkinakis: *A slant removal algorithm.* Pattern Recognition, 33:1261–1262, 2000.

[KFK01] Ergina Kavallieratou, Nikos Fakotakis, and G Kokkinakis: *Slant estimation algorithm for OCR systems.* Pattern Recognition, 34(12):2515–2522, 2001.

[Kur05] Ray Kurzweil: *The singularity is near : when humans transcend biology.* Viking, New York, 2005, ISBN 9780670033843.

[Lam09] Paul Lamere: *moot wins, Time Inc. loses | Music Machinery on WordPress.com*, April 2009. `http://musicmachinery.com/2009/04/27/moot-wins-time-inc-loses/`, visited on 25.11.2013.

[LSA94] Su Liang, M. Shridhar, and M. Ahmadi: *Segmentation of touching characters in printed document recognition.* Pattern Recognition, 27(6):825–840, 1994.

[reC] Google reCaptcha: *reCAPTCHA: Stop Spam, Read Books.* `http://www.google.com/recaptcha/`, visited on 25.11.2013.

[Red] Reddit: *4chan is using google's captcha technology to get the word nigger inserted into digital books as much as possible.* `http://www.reddit.com/r/pics/comments/cygfx/4chan_is_using_googles_captcha_technology_to_get/`, visited on 25.11.2013.

[vA09] Luis von Ahn: *reCAPTCHA Blog: Why You Should Use reCAPTCHA to Protect Online Polls*, April 2009. `http://blog.recaptcha.net/2009/04/why-you-should-use-recaptcha-to-protect.html`, visited on 25.11.2013.

[vAC09] Luis von Ahn and Will Cathcart: *Official Blog: Teaching computers to read: Google acquires reCAPTCHA*, September 2009. `http://googleblog.blogspot.de/2009/09/teaching-computers-to-read-google.html`, visited on 25.11.2013.

[vBL04] Luis von Ahn, Manuel Blum, and John Langford: *Telling computers and humans apart automatically.* Communications of the ACM, 47(2):57–60, 2004.

[Vin02] Alessandro Vinciarelli: *A survey on off-line cursive word recognition.* Pattern recognition, 35(7):1433–1446, 2002.

[Weg10] Benjamin Wegener: *Security Risks for Online Services by Relying on reCAPTCHA.* Unpublished, 2010.

[Wik] Wikipedia: *Tesseract (software) - wikipedia, the free encyclopedia.* `http://en.wikipedia.org/wiki/Tesseract_(software)`, visited on 25.11.2013.

[YE07] Jeff Yan and Ahmad Salah El Ahmad: *Breaking Visual CAPTCHAs with Naive Pattern Recognition Algorithms.* Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007), pages 279–291, December 2007.

[YE08] Jeff Yan and Ahmad Salah El Ahmad: *Usability of CAPTCHAs or usability issues in CAPTCHA design.* Proceedings of the 4th symposium on Usable privacy and security - SOUPS '08, page 44, 2008.

[YE09] Jeff Yan and Ahmad Salah El Ahmad: *CAPTCHA Security. A Case Study.* IEEE Security & Privacy, (July/August):22–28, 2009.

List of Figures

2.1 Different relink.us-captchas. 6
2.2 A “share-links.biz” captcha. 7
2.3 Different solveMedia captchas. 7
2.4 The Google reCAPTCHA 8
2.5 The baseline of the “Pavigi” challenge is offset by a curve, or rather the text is set on the curve. 9
2.6 Two possibilities for offsetting a text on a curve. 9
2.7 Slanting of the word “hersoved”. 9
2.8 Interconnection of all characters. 10
2.9 Double connected characters lead to intermediate holes. 10
2.10 An old version of the Google reCaptcha system. 10

3.1 Separation of challenge part and pro-Bono part. 11
3.2 The simple desloping process. 12
3.3 The simple desloping process sometimes runs into problems. 12
3.4 The more sophisticated approach by keeping down to the strongest plateau at the edges. 13
3.5 Selecting the plateau to stick to is not trivial. 13
3.6 Problems in selecting the deepest of the longest plateaus. 13
3.7 The slanted and deslanted versions of the “hersoved”-captcha. 15
3.8 Simulation of different shearing angles affecting the horizontal projection. 15
3.9 Results of the word-processing algorithms. 17

4.1 Parts of the challenges are already separated, i.e. not connected. 18
4.2 After initial labelling of connected regions, the algorithm detects whether single segments belong to another (bigger) group. 19
4.3 Degraded Characters 19
4.4 The ’B’ blocks the caliper from measuring the lean connection between ’e’ and ’B’. 20
4.5 Horizontal projection of the ’Bblicie’-Captcha and the detected cutting-positions hypotheses. 21
4.6 Potential cutting hypothesis of the ’Bblicie’-Captcha found by the algorithm. . . . 22

5.1 Reconstructed ’F’ by different numbers of Fourier descriptors. 25

5.2 Comparison of distorted and reference capital F. 25
5.3 Viterbi Algorithm 28
5.4 The oversegmentation chops off very small serifs at the beginning of words. 28
5.5 Arch-Detection by sending in probes from all directions. 29
5.6 Adjacent characters might form an intermediate hole, if connected at two points. . 30
5.7 Distorted times test set examples 33
5.8 Confusion matrices Tesseract↔Algorithm for distorted times test set 33
5.9 Captcha test set examples 34
5.10 Confusion matrices Tesseract↔Algorithm for captcha test set 35
5.11 Confusion matrices algorithm↔Finereader captcha test set 36
5.12 Confusion matrices 16↔32 Fourier Descriptors 36

6.1 Viterbi Algorithm 38
6.2 Final backtracked cutting positions for the 'Bblicie'-captcha. 39
6.3 Selected, processed and solved examples from the captcha test set. 43
6.4 New version of the reCaptcha system as of fall 2013. 45
6.5 The new, hardened version of the old system as of fall 2013. 45

7.1 Letter frequencies for English compared to the letter frequencies from all captchas. 48
7.2 The top 25 bigrams from the generated English dictionary compared to the corresponding frequencies of all captchas. 48
7.3 New version of the reCaptcha system as of fall 2013. 49
7.4 Deciphered number-captchas of the new reCaptcha system. 50
7.5 Problems in solving the number-captchas. 50
7.6 Deciphered Microsoft captchas. 51

A.1 Different operations applied to the original complex contour $a(n)$ and the result $\tilde{a}(n)$. 55
A.2 Curve reversal $\tilde{a}(n)$ of the contour $a(n)$. 56
A.3 Complex conjugation $\tilde{a}(n)$ of the original contour $a(n)$. 57
A.4 Both objects share the same magnitude spectrum. 58

List of Tables

3.1 Manual evaluation results of the desloping- and deslanting-algorithm. 17

5.1 First 16 Fourier descriptors for several distorted 'd'. 26
5.2 Comparison of the Fourier descriptors for the letters 'b','d','p' and 'q'. 27
5.3 Necessary orientation conditions for characters. 29
5.4 Necessary weight conditions for characters. 30
5.5 Necessary hole conditions for characters. 30

6.1 Human results for the unprocessed challenges. 40
6.2 Evaluation results of Abbyy Finereader 11 and Tesseract 3.02 for unprocessed test set. 41
6.3 Evaluation results of Finereader and Tesseract for the preprocessed test set. 42
6.4 Final evaluation results of the proposed algorithm. 43
6.5 Final verification results. 46

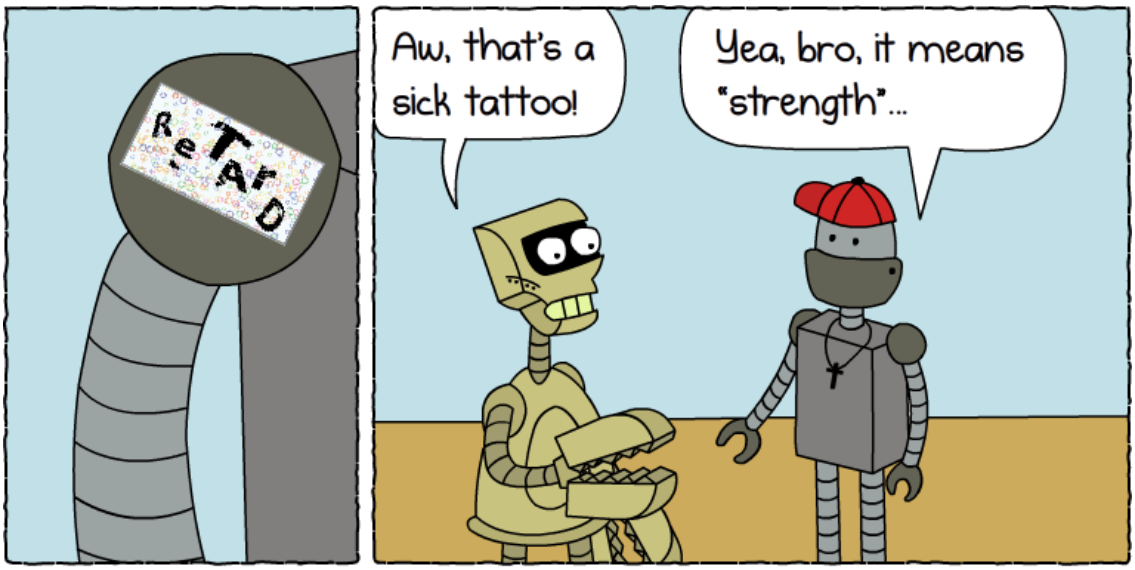
B.1 Confusion matrix of distorted times test set for proposed algorithm using 32 FD. . 60
B.2 Confusion matrix of distorted times test set for proposed algorithm using 16 FD. . 61
B.3 Confusion matrix of distorted times test set for Tesseract. (in %). 62
B.4 Confusion matrix of captcha test set for Finereader 11. (in %). 63
B.5 Confusion matrix of captcha test set for Tesseract. (in %). 64
B.6 Confusion matrix of captcha test set for proposed algorithm utilizing 32 FD. . . . 65
B.7 Confusion matrix of captcha test set for proposed algorithm utilizing 16 FD. . . . 66

Declaration of Authorship

Hereby I, Alexander Motzek, declare, according to §24(5) ASPO, that this work is, to the best of my knowledge and belief, original and the result of my own investigations, except as acknowledged, and has not been submitted, either in part or whole, for a degree at this or any other university. Formulations and ideas taken from other sources are cited as such. This thesis has not been published.

Hamburg, January 10, 2014
Place, Date

Signature



[1] http://myapokalips.com/show/23