



**FACULTY
OF MATHEMATICS
AND PHYSICS**
Charles University

BACHELOR THESIS

Lucia Tódová

OCR for tabular data

Department of Software Engineering

Supervisor of the bachelor thesis: Mgr. Miroslav Kratochvíl

Study programme: Computer Science

Study branch: Programming and Software Systems

Prague 2019

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In date

signature of the author

Dedication.

Title: OCR for tabular data

Author: Lucia Tódová

Department: Department of Software Engineering

Supervisor: Mgr. Miroslav Kratochvíl, Department of Software Engineering

Abstract: Abstract.

Keywords: key words

Contents

Introduction	2
1 Text Recognition Algorithms	3
1.1 Preprocessing	4
1.1.1 Scaling	4
1.1.2 Contrast enhancement	6
1.1.3 Binarization	7
1.1.4 Deskew	8
1.1.5 Noise Reduction	10
1.1.6 Scanning border Reduction	11
1.2 Basic Techniques	12
1.2.1 Page Segmentation	12
1.2.2 Representation and Feature Extraction	14
1.2.3 Character Classification	15
1.2.4 Deep Learning	16
1.3 Available Implementations	16
1.3.1 Tesseract	17
1.3.2 OpenCV	17
1.3.3 ImageMagick	18
1.3.4 Commercial Software	18
2 Layout Recognition For Tabular Data	20
3 Table Recognition Implementation	21
4 Results	22
Conclusion	23
Bibliography	24
List of Figures	27
List of Tables	28
List of Abbreviations	29
A Attachments	30
A.1 First Attachment	30

Introduction

Over the last few years, the importance and usage of digital documents has grown rapidly. Today, almost everything can be found in digital form. However, there are still tons of documents that need to undergo the process of digitization - conversion of analog signals or information into a digital format. Creating a digital document for viewing from an analog input is not hard - a simple scan should suffice. Complications arrive when the user wishes to modify the document. What a scan provides is merely an image, which has no information about its elements - text, paragraphs, numberings, headers/footers, forms, tables... This is the moment when an OCR software is needed.

Optical Character Recognition is a widespread technology used for converting images containing written text into a machine-readable text data. It became popular during early 1990s while trying to digitize historic newspaper, and is now frequently used for data entry automation, ticket and voucher validation, tax-free shopping, automatic number plate recognition, as well as assisting blind and visually impaired people.

Different OCR softwares use different techniques to achieve their goal. They mostly consist of a variety of combined heuristic algorithms with an equally important part of preprocessing. Although most of these softwares guarantee their accuracy rate to be above 98%, the measurement techniques are not always the same and therefore the rates can be misleading. Furthermore, most of these softwares already work on different kinds of assumptions about documents - that they must have a specific layout, contain a header and a footer, have all text lines horizontally aligned et cetera...

In this thesis we specifically focus on recognition of tabular data and the problems that come along with it. We describe the possibilities and

```
// co je digitalizacia, je popularna, potrebujeme zistit viac ako len obrazok,  
treba informaciu o dokumente  
// co robia OCR softwares  
// ake su problemy s OCR softwares  
// my konkretne riesime tabulky  
// thesis bude mat nasledovnu strukturu
```

Related Work

1. Text Recognition Algorithms

The root of any complex OCR software is text recognition. Although this is easily done in printed documents, there can be a lot of factors that complicate this process. As described by Bieniecki et al. [2007], the few of the most important problems were found to be:

- **different fonts** - Nowadays, there exist a number of fonts and styles. To hold the information about each and every one would be unsustainable for any OCR software. Therefore, it needs to make assumptions and use heuristics to match its vision of the character with the one it is reading.
- **handwritten documents** - The software has no notion about whether the document was handwritten or not. It sees the characters as merely a more complicated font. On contrary to the font, the characters that are supposed to be the same do not always have to look that way. This is why the OCR software has to count on minor character mistakes. These depend on different factors - line height, length, character size... Furthermore, handwriting can sometimes be unrecognizable even by a human eye. This case often prompts the software to fail.
- **low quality of the scanned document** - This can be caused by a poor scan or just a poor initial image and mean a variety of things. The image can be low-contrasted, not sharp enough, lines can be disrupted or pixelated (mostly in case of a low scan DPI), it can contain a lot of noise... These problems are usually partly solved during the preprocessing part of OCR algorithm.
- **skew problems** - These problems also fall under the part of preprocessing. As the result of a scan or even a photography is almost never an image with a perfectly aligned text as it was on the paper, this needs to be fixed, so the actual algorithm can count on correctly aligned images.
- **colors** - The general rule is - the better the contrast of the image, the better the OCR results. Colored images generally have a lower contrast than when they are in black and white. Before passing to OCR, they therefore undergo binarization.
- **similar characters** - Even people sometimes make the mistakes when distinguishing characters like S and 5, O and 0, or I and l. OCR software can often make the same mistake, especially when different, special fonts are used, or in case of handling handwriting.
- **inconsistent font use** - To determine word spacing, most OCR engines calculate gaps between characters and heuristically decide which characters belong together. With different fonts, spaces also differ, which may produce unsatisfactory results.

These and multiple other factors are also the reason why many OCR softwares work on "assumptions" about documents - that the document has only one

column, is not handwritten, that its lines are horizontally aligned et cetera. Also, this is why no OCR software can work perfectly and will always keep encountering documents that are "unrecoverable".

However, there are many ways in which we can improve the results of the OCR.

1.1 Preprocessing

Many of the existing OCR engines already come with some kind of built-in pre-processing filter. Their problem is that they most likely do not match the exact image case and are usually very simple.

This is why the best practice is to firstly preprocess the image and then pass the result to the OCR algorithm.

In this section, we will discuss the most important image transformation that we can perform to improve the results of the OCR.

1.1.1 Scaling

OCR usually benefits from a properly scaled image with good resolution. It contains more detailed information that can be used to interpret text, which leads to higher accuracy of the result. This can be specifically applied to the case of resolution and bit-depth.

Many popular OCR engines (Tesseract [Packard, 1996], ABBYY FineReader [ABBYY]) encourage their users to use a 300 DPI images. Their reason for that is pretty simple - it is the point where you gain the most accuracy without sacrificing speed and file size. After running a few tests for the same image with different resolution on OCR engines [Bieniecki et al., 2007], the improvement gap between 200 DPI scan and 300 DPI scan is at least 2 times the improvement gap of any other resolutions. Also, when comparing other images above 300 DPI with a 300 DPI image, the improvement gap is nearly absent. It is obviously still there, but in almost all cases, the higher DPI is not worth the time and space.

Another reason for the use of 300 DPI is the fact that most OCR engines are trained to this resolution. Some engines (like the already mentioned ABBYY FineReader [ABBYY]), no matter what resolution you give them will actually sample up or down to get to 300 DPI.

Achieving a 300 DPI by a scan is simple enough. If the user has already provided us with an image of a different resolution, our next steps depend on what the resolution is:

DPI of new image < 300 : If we were to just scale the image, the result would be pixelated, blurred and not sharp enough. The most affected parts would be the diagonal edges of elements. They would appear jagged (*aliased*), which gives the image overall a poor quality. To minimize this unwanted effect, a technique called *interpolation* is used.

Interpolation works by using known data to estimate values at unknown points. It specifically approximates the resulting pixel's color and intensity based on the values at surrounding pixels. It therefore already includes the

process of *anti-aliasing* (process used to minimize aliases), as it is based on the same technique.

The more data we have, the better the interpolation - therefore we will still see the difference between a resized image from 72 DPI to 300 DPI and 200 DPI to 300 DPI.

Interpolation algorithms can be grouped into two categories: *adaptive* and *non-adaptive*. Non-adaptive methods treat all pixels equally, while adaptive methods change depending on what they are interpolating, specifically smooth texture vs. sharp edges. Adaptive methods are primarily designed to minimize the presence of interpolation artifacts in regions where they are most apparent, and they differ by the way they detect edges.

Non-adaptive methods do not distinct different pixels. Their complexity depends on the number of adjacent pixels during interpolation, which is also the criterion by which the existing methods are divided [Fadnavis, 2014].

- **Nearest Neighbor Interpolation:** This algorithm considers only one pixel - the closest one to the interpolated point.
- **Bilinear Interpolation** considers the closest 2x2 neighborhood of known pixel values surrounding the unknown pixel. It then assigns the weighted average of these 4 pixels to the final interpolated value.
- **Bicubic Interpolation** goes one step beyond bilinear and takes the closest 4x4 neighbours, which sums up to the total of 16 pixels. During the calculation of the final value, pixels closer to the interpolated point are given a higher weighting.

There also exist higher order interpolations, such as *Spline*, *Sinc*, *Lanczos*, *Discrete Wavelet Transforms* [Fadnavis, 2014]... They take more surrounding pixels into consideration and calculate the resulting value through more complex functions, like transforms or sampling functions. The results are more accurate than those of simple calculations, but the process takes up significantly more time. During OCR, such complex and time-consuming functions are not necessary, as we are not yet working with any rotations and distortions and just need to resize the existing image.

For the best quality/time ratio, the popular decision in most cases is *Bicubic Interpolation*. Although *Nearest Neighbor* and *Bilinear* methods are extremely fast, the results were found to be poor [Han, 2013].

DPI of new image > 300 : In this case, we need to solve the exact opposite problem as previously - decrease the number of pixels and decide what will the values of the new ones be. The easiest approach would be to "pick every nth pixel", but the same problems as previously arise - blurring, aliasing, pixelation...

The simplest and most widely used approaches are very similar to what is already explained above. To calculate the value of the resulting pixel, we choose a number of surrounding pixels. We calculate their weighted

average and assign this value to the unknown pixel. As previously, the 4x4 neighborhood turned out to have the best quality/time ratio.

There are obviously many other ways that our desired result can be achieved. Already mentioned *Spline*, *Sinc*, *Lanczos* and other "reverse interpolations" are a few of them. Worth noting are also *Fourier transformation*, *perceptual based methods*, *content-adaptive methods* and other adaptive and heuristic techniques. Many of these methods produce even better results, but, as mentioned before, most preprocessing engines still stick to the simpler "reversed Bicubic Interpolation" in a successful attempt to speed up the algorithm.

1.1.2 Contrast enhancement

Another important factor that results in low-quality images is low contrast. Even human eye has a harder time recognizing images with this element, and the same goes for OCR engines.

An image *histogram* is used to improve this aspect. It is an accurate representation of the distribution of data and, in case of image histograms, it represents the tonal distribution of an image - x-axis in histogram stands for all available tonal levels, and y-axis represents the number of pixels for each tonal level.

Grayscale images are represented by only one histogram, where the x-axis contains all the available grey values. However, in colored (RGB) images, the histogram is displayed by the terms of three separate histograms - one for each color component (R, G and B). Increasing contrast for colored pictures is therefore a more complex task. This, however, is not a case we need to solve, as the next part of preprocessing is binarization. As we will mention later, binarization works better on grayscale images. Therefore, the conversion to grayscale can already be performed in this section. Only after that, the contrast enhancement will take place.

Grayscale conversion is the process of computing and assigning the grey value for each colored pixel from its attributes. Over the years, different approaches have been created. That is why, when using image editing software such as Adobe Photoshop or GIMP, there are many options available for grayscale conversion, each of them producing slightly different results.

The most simple approach is to find the average of the R, G and B values. However, this does not really preserve the brightness of the image. Therefore, corrections like *luma* [Kanan and Cottrell, 2012] are used. This algorithm is based on the fact that humans perceive different colors differently (green more strongly than red, and red more strongly than blue), therefore, each color is weighted dependant on how it is perceived by the human eye. Another way to approach grayscale conversion is to represent the color based on its HSL values and convert it to its least saturated variant.

As described by [Čadík, 2008], there exist many more algorithms approaching this problem. Most of them require more complicated computations and try to preserve attributes of the image that may be lost during the process - like contrast or luminance.

Now that we have a grayscale image, the next step is *contrast enhancement* with the help of already mentioned histograms. In them, contrast is represented

as distribution of pixels. The more evenly the pixel counts cover a broad range of grayscale levels, the better the contrast of the image. Pixel counts that are restricted to a smaller range indicate low contrast.

To achieve an image with a higher contrast, we therefore need to stretch its histogram. In the following paragraph, we will briefly explain the basis of a few of the most popular methods. However, many more complex and complicated techniques already exist, such as *CLAHE*, *Dynamic Stochastic Resonance* or *transforms* (Fourier, Discrete Cosine, Wavelet)... [Suganya et al., 2013]

- **Linear Stretching** method linearly expands the original digital values of the histogram into a new distribution. There exist three methods of linear stretching - *Min-Max*, *Percentage* and *Piecewise Contrast Stretch* Al-amri et al. [2010]. All these methods make subtle variations within the image data more obvious, are best applied to remotely sensed images with Gaussian or near-Gaussian histograms.
- **Non-linear Stretching** is done by functions with non-linear form. These include, for example, *logarithmic transform*, *exponential transform* or *Power Law* [Al-amri et al., 2010]. The downside of these methods is that each value in the input image can have several values in the output image - therefore, the objects in the original scene lose their correct relative brightness value.
- **Histogram Equalization** is one of the most popular methods of non-linear stretching. This method is based on transforms dependant on the probability distribution function of the histogram [Dorothy et al., 2015]. Although it produces unrealistic effects in photographs, it is widely used for scientific images.

1.1.3 Binarization

As already mentioned, element detection in color, or even greyscale images is a way more complicated task than with monochrome one (1 bit) images. This is due to the the lack of contrast and therefore possible blending of different edges during recognition.

Binarization is a process that converts an image to its monochrome one form. This means that every pixel on the binarized image will be either black or white.

Many binarization algorithms work under the assumption that the input image is already in greyscale. If not, they firstly transform the image to greyscale, then apply the binarization techniques. This does not concern us, though. In the last section, where we talked about contrast enhancement of the image, our results were already in greyscale. Therefore, for the next part, let us assume that the images we work with are strictly in greyscale.

The simplest approach to binarization is *global thresholding* [Lund et al., 2013]. It works by choosing a threshold value and iterating over the whole image, comparing the value of each pixel to this threshold. If it is greater, the pixel in resulting image will be black, otherwise white. As the brightness and contrast of different images vary, it is impossible to choose a threshold suitable for all images.

Therefore, other more complex methods are used:

- **Otsu’s method** [Yousefi, 2011] is one of the most widely used and mentioned methods for image binarization. First, it computes the threshold value from the input image and then applies global thresholding. To get the optimal value of the threshold, Otsu’s method works by dividing the pixels into two groups - background and foreground pixels. It chooses these groups so that the within-group variance is minimized and between-group variance is maximized. Upside to this method is that it is simple and easy to implement. However, upon applying it to images with uneven dark or bright spots, it seems to be unusable.
- **Local Thresholding:** [Sezgin and Sankur, 2004] In contrast to Otsu’s method, where all pixels behaved regarding to their common threshold value, local thresholding methods assign different segments of the image different threshold value. This solves the problem with unevenly lit spots - darker spots will be assigned a lower threshold value, while brighter will be assigned a higher threshold value. The difference between different local thresholding algorithms is the way the values are assigned. For example, *Niblack’s Method* determines the threshold of every pixel by using the mean and standard deviation of surrounding pixels, which causes errors in blank regions of the image. *Sauvola Method* improves this by using the dynamic range of image gray-value standard deviation. However, this often results in text thinning. Another method worth mentioning is *Bernsen’s*, where the threshold is set to the mean value between the lowest and highest gray-level pixel in the neighborhood.
- **Adaptive methods:** Every document is different. Some techniques might work for a few and corrupt others. To determine which technique to use for each document image would be a useful feature in most cases. This is the exact problem that adaptive methods try to solve. As described in [Sauvola and Pietikäinen, 2000], their approach is to analyze the document image surface and determine the most useful method with the help of a ”hybrid switching” module.

Different kinds of improvements have been created in the form of new methods. They mostly consist of combinations of already mentioned methods, or have a similar basis. Further exploration of possibilities was, for example, done by Sezgin and Sankur [2004].

1.1.4 Deskew

Skewed images are probably the most common problem that many OCR engines struggle with. They usually appear when an image is scanned or photographed crookedly.

One of the steps of OCR algorithms is *page segmentation*. It is the process that decomposes a document image into elements. This division is based on vertically and horizontally aligned characters, spaces and other elements. If the documents have Manhattan layout, decomposition is even easier and can be done by vertical and horizontal cuts. We will discuss this topic thoroughly in the following chapter. For now, the important thing to know is that skewed images pass misaligned data

to the OCR algorithm. This causes more errors and ultimately leads to poorer results.

There are different approaches that deal with this problem. All of these methods work (for more accurate results) after binarization of the image and generally assume the skew angle to be no more than 15° . Algorithms that work on greater skew angles also exist [Rosner et al., 2014a]. They are not as widely used, though, as most of the time, correction needs to be done on scanned documents - which are mostly only slightly tweaked.

As described by Rosner et al. [2014b], the most effective deskewing algorithms include the following:

- **Hough Transform:** This method is used to find straight lines in an image and given their rotation, determine the skew angle of the whole documents. It is based on a feature extraction technique called *Hugh Transform* [Shukla et al., 2016]. This method is applied on different lines. Then, it determines which line is the most represented in the input image by comparing their number of pixels. Only thing left is the computation of the resulting angle. Although highly effective, computations of transform can be quite time consuming. This is why the transform is usually not done on every point of the image, but only a set of chosen input points. Another problem occurs if pictures or other graphic data are present in the input image (as Hugh Transform counts with aligned pixels). To solve this, we only choose input points from the text data.
- **Projection Profile:** Projection Profile method is based on creating a histogram of black pixels along horizontal or vertical scan lines. For an image with horizontal scan lines, this histogram will have peaks at text line positions. The algorithm estimates the skew by rotating the image in different angles and comparing the resulting histograms. The one with the most peaks and valleys belongs to our final angle. The problem with this technique is that it works on text areas only, as images or other elements can disrupt the histograms.
- **Cross Corelation** works similarly to Projection Profile method mentioned above. Firstly, it selects a number of vertical segments from the image. It then performs the Projection Profile algorithm on these segments and by comparing then, it can estimate the final skew angle. The drawbacks of this method are similar to those of Projection Profile. However, it usually needs to process less data with no loss in accuracy, given a good choice for the width of the columns.
- **Connected Component Clustering** - or *Nearest Neighbor Clustering*. Thoroughly described in Lu and Tan [2003], these algorithms are based on finding coherency between connected components of the image. They either compute the skew of every component (using their bottom line) and compare the results, or group components by neighbours and create a group histogram for examination or many other more.

Worth noting are also other methods like *Fourier* [Singh and Kaur, 2013], *Wavelet Decomposition*, *Radon Transform*, *PCP*, *one-pass* or *multi-pass* skew correction, *morphology algorithms* etc... [Rosner et al., 2014b].

1.1.5 Noise Reduction

Image noise is an usually unwanted random variation of brightness or color information in images, reminding film grain in digital cameras. It is often caused by the sensor of a scanner or a digital camera attempting to record small amounts of light.

For OCR engines, processing noise can be quite tricky. Its presence is especially crucial during binarization and edge detection. These processes are based on the differences between the "outside" of the element and its "inside". When noise is present, it can disrupt these borders, which can often cause the failure of recognition.

Based on the way they originate and are processed, we concentrate on these types of noise:

- **salt and pepper noise**, also known as *impulse noise*, can be recognized as randomly occurring black and white pixels. It is caused by sharp and sudden disturbances in the image signal, and can be most likely seen on old photographs.
- **statistical noise** - is an unexplained variability within an image. It is characterized by a probability density function. Most common is *Gaussian noise*, *Shot noise*, *Quantization noise* or film grain.
- **other** - like *anisotropic* or *periodic* noise. These are often found on old photographs or documents.

There exist various approaches dealing with noise reduction. These include different types of filtering or comparison methods to firstly detect which pixels were corrupted by noise and then determining their value in the output image.

One of the most commonly used techniques, found also in image editing software like Adobe Photoshop or GIMP, are **linear filters** [Mythili and Kavitha, 2011], like *mean filter*, *Gaussian filter* or *Wiener filter*. They work on averaging the color values of neighbour pixels, so they can help assure an averaged view of the contrast along regions of different color. This, however, rather than correcting the color of the affected pixel, results in creating an area with an average (and incorrect) color. This blurs the edges and smooths out other fine details. They are still widely used, though, as they require only a small amount of computations, which leads to increased speed and easy and simple implementations.

Other filters used for image denoising include **non-linear filters** [Mythili and Kavitha, 2011]. The most popular of them is *median filter*. It works similarly like a mean filter, but for each pixel, instead of replacing its value with a mean of surrounding pixels, it calculates the median. In comparison with linear filters, it provides considerably sharper images and in cases preserves the edges. The downside is the increased run time, and with high level of noise, results are almost the same as with linear filters. However, it is highly effective on salt and pepper noise.

Worth mentioning is also **non-local means method** [Buades et al., 2005]. It is based on taking a mean value of all pixels of the image, weighted by how similar these pixels are to the target pixel. This technique produces an even sharper and clearer image results as mentioned filters. However, it comes with the expense of additional increased speed.

What combination of techniques to use for the optimal results greatly depends on the input image - if there is little to no noise, de-noising an image can cause more harm than good. Therefore, it is always important to approximately compute the signal:noise ratio before applying any kind of filters.

To calculate this properly is a more complex task than thought. Although simple algorithms have already existed for a few years, they return poor results and most of them are almost not worth using because of the time complexity. They are mostly based on comparison - testing the difference between each pixel and its neighbors for various threshold numbers based on the dynamic range and calculating deviation, picking "windows" of different sizes and calculating their means and deviation, based on which we decide whether the center pixel is corrupted or not etc...

For a more complex (and sufficient) solution, we have to run many more mathematical calculations on the input image [Maragatham et al., 2015]. These calculations require a lot of knowledge in the field of probability and statistics, specifically probability distribution. The basic idea for these noise recognition algorithms is *separating frequencies*. This is usually done by dividing the image into blocks and running a transformation (for example a *discrete cosine transformation*) on each of the blocks. The frequency components are then grouped by increasing frequency. Then, the extraction of statistical features (such as kurtosis, mean, standard deviation) is performed.

Noise is most likely to be found on higher frequencies and we use this observation along with our computed statistical features to determine the thresholding on higher frequencies. If the threshold of the input image is greater, the image is determined to be noise-free.

1.1.6 Scanning border Reduction

Scanned documents often have visible dark borders (*marginal noise*). These are created either during scanning (due to the presence of neighbouring pages) or are an unwanted result of the binarization process.

Similarly to noise reduction, running this step could only be harmful if there is no marginal noise present. Therefore, the algorithm has to have two steps - *marginal noise detection* and, if successful, *marginal noise reduction*.

Marginal noise is present as a dark connected component of the page. This is a fact that most detection algorithms take advantage of. Firstly, they extract the connected components of the page. This can be done by a *black filter* (a moving rectangular window that calculates the ratio of black pixels on border [Shafait and Breuel, 2009]). After obtaining the connected components, different heuristics are used - most based on removing the connected components near the edges. In the end, usually a "cleanup" is performed (like *white filter* [Shafait and Breuel, 2009]) - for removing all unnecessary elements around the borders (an unwanted text from the neighbour page, noisy components in the corners...). The

results of this algorithm depend greatly on the values of thresholds, margins and other hardcoded parameters. Different images work well under different values. This is why the cleanup part's parameters are usually dependant of the results of previous parts.

Another approach is the one mentioned by Peerawit and Kawtrakul [2004]. It is based on edge density calculations, and the fact that text areas have generally a much lower density than edge areas. An edge detector (in this case, *sobel*) is used for examining vertical marginal noise, and later horizontal marginal noise. It returns the positions of the margins. Anything beyond these positions is detected as an unwanted noise, and can be therefore removed by filters or other removal algorithms.

1.2 Basic Techniques

Nowadays, two approaches are proposed for text detection in OCR. The first one is an approach based on heuristics - page segmentation, line and element detection and only after that, character detection, which is based on different approximation and sometimes naive algorithms. The second one is an approach that is widely used and researched today - and that is the use of neural networks.

In this paper, we will mostly be concentrating on the heuristics approach of the OCR problem. However, deep learning also deserves a mention and we will also briefly discuss this method in this section.

A lot of OCR engines nowadays differ in the way their heuristic approaches work. They all agree on one thing, though, and that is the succession of steps that need to be implemented. Firstly, a page segmentation has to be performed. This divides the pages for determining the elements of the image. Now, the recognition algorithm could simply take place. However, most of the existing OCR engines (like Tesseract) "pre-process" the output of page segmentation. They use steps like *representation* and *feature extraction*, which are based on simplifying the extracted elements of image documents. This contributes to better results of the OCR, and, last but not least, helps to improve the speed of the algorithm.

Only after this, the character recognition algorithm is applied.

We will go over these steps in the following individual subsections.

1.2.1 Page Segmentation

In many cases, OCR system heavily depends on the accuracy of page segmentation process [Mao and Kanungo, 2001]. There are three main approaches [Shafait et al., 2008] to this problem - either we start with an image as a whole and recursively divide it into parts until criterion is met, or we start with individual pixels and group them according to similarity of their attributes, or we use a combination of these two methods. Based on these methods, there exist a few concrete algorithms widely used for heuristic page segmentation.

To start off, we are going to have a look at the **X-Y cut** algorithm, also referred to as *RXYC algorithm*. It is a tree-based algorithm, where the node of the tree presents the whole image page and leafs of the tree the individual segments. Algorithm starts by splitting the root of the tree into

rectangular parts. Every step of the recursive algorithm includes projection profile and threshold computations, depending on which the node will either be declared to be the end segment, or a split into two other rectangular parts will be executed. The algorithm finishes once there are no more nodes to split.

Although this algorithm is easy to implement and pretty fast, according to the benchmarking test done by Shafait et al. [2008], it should not be used on other images than clean, properly skewed, with no noise whatsoever. In the presence of any imperfections, it usually takes the whole image as a segment.

Slightly better results are achieved by the **smearing** algorithm, also referred to as run-length smearing algorithm (RLSA). It is based on linking together black areas that are no more than C (which is a predefined constant) pixels away from each other. This algorithm is applied both row-wise and column-wise, and the resulting bitmaps are then combined. Although this algorithm works only on binarized images, we do not need to handle this problem. At this part of the algorithm, all passed images should be binarized due to preprocessing steps.

What affects us, though, is that this algorithm classifies text-lines merged with noise blocks as non-text. So although its results turn out better than those of RXYC algorithm, it is still not usable for classical text-documents. However, it is widely popular among vehicle plate recognition.

Other improvements were made by using the **Voronoi-diagram based algorithm**. This technique firstly extracts connected components of the image and, from sample points of the edges of individual components, then creates a Voronoi diagram. A "clean up" of the diagram then needs to be performed - which means deleting the edges that pass through a connected component, or the edges with its associating connected components that have a small distance or a too large area ratio.

As already mentioned, this method provides better results on text documents, even the ones with noise. However, it does not work well on images with different fonts or styles, as the spacing is estimated from the document image. This results in over-segmentation errors and it is advised to use this method on a homogeneous collection of documents.

Another algorithm also based on extraction of connected components is the **Docstrum** algorithm. However, instead of creating a diagram, it performs a near-neighbor clustering. We already mentioned a similar approach in [deskew] section, and this algorithm also depends on the skew of the image. Firstly, connected components are divided into two groups - one with characters of the dominant font size, the other with characters from titles or section headings. For each connected component, its nearest neighbours are found. The algorithm then determines a histogram of distances and angles between nearest neighbors. Peak of the angle represents the dominant skew, which is used for determination of a within-line nearest neighbor pairs. These pairs are then merged into text-lines, which are merged into blocks.

The downside to this approach is similar to the one in Voronoi diagram, and that is spacing variation. Due to different font sizes and styles, this can lead to over-segmentation. Therefore, the Docstrum algorithm is also advised to use on a homogeneous collection of documents.

For dealing with heterogeneous documents, **whitespace analysis** is a better choice. This algorithm is based on the assumption of white background. It firstly tries to cover the the background with the union of white rectangles. These covers are then sorted in respect to a sort key based on a weighting function, which has the purpose of assigning higher weight to taller and longer blocks. Then, covers are added to form background cover until the sort key is less than the threshold.

Most of the time, whitespace analysis provides satisfactory results. In our case, we do not even need to handle the white background assumption, as we are already working with only binarized images. However, it is very sensitive to the stopping threshold, therefore results in over-segmentation or under-segmentation.

Similar approach to whitespace analysis is **constrained text-line detection** approach. In contrast, it starts with finding the whitespace rectangles that cover the page background in order of decreasing area. From these rectangles, columns separators are chosen based on their properties - aspect ratio, width, and proximity to text-sized connected components.

The upside of this approach is that it is nearly parameter free - therefore, it is a good option for documents with different font sizes or layouts.

There exist many segmentation methods that approach this problem differently. For example, with segmentation of scanned books which mostly have Manhattan layout, a simple deskewing and horizontal and vertical cuts should suffice. Also, algorithms focusing on recipe or ticket analysis already know what the approximate layout of the received image will be. They can therefore simplify these complicated algorithms for their purposes.

1.2.2 Representation and Feature Extraction

Upon extracting elements by page segmentation, we are left with an unnecessary amount of information. We could feed all of them into a recognizer. This would, however, decrease accuracy of recognition and significantly increase its time complexity. For these reasons, a set of features is extracted for each element that distinguishes it from other classes while keeping characteristic differences. This can be done in various ways [Chaudhuri et al., 2017].

The first group of techniques we will discuss are the **global transformation and series expansion** methods. They include methods like *Fourier* or *Gabor* transform, *Fourier Descriptor*, *Wavelets*, *Moments* and *Karhunen-Loeve expansion*, and are based on the representation of continuous signals by a linear combination of simpler functions. The coefficients of the linear combination provide a compact encoding known as transformation or series expansion. Deformations like translation and rotations are invariant under global transformation and series expansion.

Another representation and following extraction technique is **statistical representation**. Unlike transforms, it uses a statistical distribution of points to take care of font and style variations to some extent. Methods based on this representation are e.g. *zoning, crossings and distances* and *projections*.

Various character can also be represented by their geometrical and topological features, based on the basic line types that form the character skeleton. The output of this representation and feature extraction process can be assigned into a feature vector. This is called **geometrical and topological representation**, and includes techniques like *extracting and counting topological structures, measuring and approximating the geometrical properties, coding and graphs and trees*.

To sum up, the goal of any representation and feature extraction technique is to create a "skeleton" for each character by selecting the most representative information from the raw data which maximizes the recognition rate with the least amount of elements. However, there are still a number of differences between the features. The most noticeable difference is reconstructability - for some methods, the image can be reconstructed to its previous version solely because of features. Others, like statistical representation, lose information about the original image and would need complicated approximation algorithms to restore at least some of it. Another difference is the transformations to which features are invariant.

1.2.3 Character Classification

OCR engines widely use the methodologies of pattern recognition, which assigns an unknown sample into one of its predefined classes. There exist various approaches dealing with these methods, which are not necessarily independent of each other. Most often than not, OCR engines combine multiple methods to achieve the most accurate results.

The first and most simple way to approach classification is by **Template Matching** [Muda et al., 2007]. It is based on the existence of predefined *templates* - multiple bitmaps containing characters of the alphabet. Improved version of this method have an extended database of templates, including numbers and special characters. Once a character is detected, it is passed to the algorithm and for every existing template, its similarity ratio is calculated. The template with the greatest ratio is then assumed to be the recognized character. This method has various implementation depending on how the ratio is calculated - for example *cross correlation, normalized correlation* or *euclidean formula* can be used.

Although the implementation of this method is very simple, even small disfigurements and noise can greatly affect its efficiency. Also, in this case, a feature extraction would be unnecessary as all templates are created manually.

To make use of the previous steps, **statistical techniques** [Dongre and Mankar, 2011] come in handy. They are based on statistical modeling of the data. To determine the output, extraction of the features from input image (such as size, shape, intensity) is first performed. Then, with the help of statistical decision functions, these features are compared to the statistical model.

The problem with these algorithms is that they have no information about whole-part relations. For this reason, a newer approach has been tested out over

the past few years - *machine learning* [Dongre and Mankar, 2011].

Machine Learning is a method of data analysis based on artificial intelligence. Over time, it builds models of "training data". Based on them, it makes decisions and predictions on its input.

In the case of character classification, training data is created by passing various characters into the engine and also providing it with the correct output. In this way, the engine learns how different characters should look and applies this knowledge to its input.

This method is for example used in a one of the simplest machine learning algorithms, and that is the **KNN Classification Algorithm**, or K-Nearest Neighbor algorithm. The input image is compared to the training data and chooses its K nearest objects (objects that are the most similar). After that, the image is classified with being assigned to the class most common among its K nearest neighbors.

More complex methods include the **Support Vector Machine algorithm** (SVM). This approach divides the data received into two classes - training and testing data. The goal of SVM technique is to deliver a model that predicts the output of the test set.

The learning is done by a *SVM kernel*. The task of the kernel is to take the input data and transform it into the desired form using a combination of mathematical functions. Other than kernel functions, there are other parameters that tune the output of the SVM algorithm. *Margin* parameter tells the engine the separation of line to the closest class points. *Gamma* parameter decides how far the influence of a single training example reaches and *regularization* parameter controls the misclassification of elements.

Many experiments have been executed in the field of machine learning for character classification. However, none of them can guarantee the accuracy of different approaches, as they all greatly depend on the training data. The more training data the machine can get, the more accurate it is.

1.2.4 Deep Learning

– TO-DO - research deep learning for other purposes, maybe for the whole OCR

1.3 Available Implementations

Over the past few years, OCR has become a part of our everyday lives. The demand for a reliable OCR engines has therefore risen, which lead to many new implementations or improvements of the already existing ones.

Most people take the expression "optical character recognition" to mean solely text recognition. This term, however, has a wider meaning - it also includes the recognition of other document elements, like images, forms, tables and many more.

There exist few OCR engines that provide all the features an OCR software should have - such as different types of preprocessing, support for all file extensions, recognition of different fonts, including handwritten documents. This is not needed in most cases. For example, customers using OCR for ticket validation do not need to process handwriting or different fonts, and customers trying

to achieve automatic number plate recognition do not need to process tables or forms or any other elements.

For this purpose, a lot of OCR engines focusing entirely on one or more cases were implemented, as it is less complicated, time consuming and does the job. However, all of these existing OCR engines, no matter how difficult, have one thing in common - they all have to recognize text elements to some extent.

In this chapter, we will go over the most popular and widely used OCR engines that focus (among other things) on word and character recognition. We will also discuss engines used for the preprocessing part of the algorithm.

1.3.1 Tesseract

Originally developed by Hewlett-Packard CO around 1990 [Packard], Tesseract is one of the most robust and accurate OCR open-source engines. When it was firstly developed, Tesseract could only accept TIFF images containing simple one-column text in English language. Since then, it has undergone a lot of improvements and added many features. As of today, Tesseract supports multi-columned documents (via its page-layout analysis), claims to support over 100 languages (including right-to-left text such as Arabic or Hebrew), works on different input and output image formats (with the help of Leptonica [Bloomberg, 2001] library) and is available for Windows, Linux and even Mac OS. In its latest version (4.0.0), Tesseract also added a new neural network (specifically a LSTM network) focused on line recognition.

Tesseract does not have a GUI and works as a command line program. It is used mostly for development purposes and provides an OCR engine (libtesseract) that gives developers a chance to create their own applications using tesseract API. Also, Tesseract contains no preprocessing algorithms. It advises users preprocess the input images themselves [Packard, 1996].

This is the reason why many wrappers and other 3rd party projects using Tesseract have been created. These projects mostly focus on creating GUIs for Tesseract or adding preprocessing algorithms to make the Tesseract engine more user-friendly. Although Tesseract is still in the process of development, it plans on doing no such thing and its future work consists mainly on focusing on LSTM networks.

Tesseract is one of the few OCR open-source engines. That is why it is so widely used for development and research purposes. However, for quick, everyday and user-friendly purposes, this is not the choice to go with.

1.3.2 OpenCV

OpenCV is an open source computer vision and machine learning software library. It claims to have more than 2500 optimized algorithms used for face detection and recognition, 3D object manipulation, photography editing (like red eye removal), tracking of moving objects or camera movements and other image processing functions [Intel Corporation, 2000].

On contrary to Tesseract, OpenCV is a user-friendly library. It is widely used as a framework for creating OCR applications and contains a lot of preprocessing functions. These can be then passed to the recognition algorithm, which makes

the whole process of recognition easier and simpler. However, its main area of expertise is not OCR and definitely not document manipulation. For recognition, it mostly applies the *template matching* technique. This technique is simple and easy to implement and works very well on ticket validation, credit card recognition or car plate recognition. Although these are all areas that OpenCV is widely used for with great success rate, the outputs from text document OCRs are poor [ref? to find]. It does not even have a library specialized for OCR.

The reason for mentioning OpenCV is exactly for its vast variety of preprocessing options. There are many projects that incorporate the work of OpenCV and Tesseract to achieve the most accurate and, most importantly, user-friendly results. Although OpenCV is mostly used via its Python interface, it also has a C++ and Java interfaces which can connect with Tesseract either directly via its C++ API, or through a wrapper like PyTesseract.

1.3.3 ImageMagick

Even though it is not used for OCR and has no feature that would allow such a thing, ImageMagick is a software worth mentioning. In case of OCR, it is used widely for the preprocessing part of the work. It is a free, open-source software suite containing numerous features useful for text image processing, like transformations, color management, large image support, format conversion (it claims to be able to process over 200 different file formats), noise reduction and many more [Cristy, 1999]. Its functionality is mainly utilized from command-line environment. ImageMagick also has various features that enable users to create and modify images via a language interface like Magick++(C++) or Python-Magick(Python), but these interfaces are not as rich as the library itself.

Although they might be used for the same purpose in case of OCR, ImageMagick and OpenCV are utterly different programs. OpenCV's goal is computer vision algorithms, like object identification and recognition, while ImageMagick is used solely for image manipulation - that is, image processing. This focus of ImageMagick leads to better accuracy of its results. The downside of this is that ImageMagick can only apply functions to images and does not make assumptions about images itself. This leads to a more complicated integration of the software with Tesseract or other OCR engine.

1.3.4 Commercial Software

On contrary to Tesseract, there exist many OCR softwares that are used for commercial purposes. Although these are practically useless for developers, they produce pretty satisfactory results, in many cases better than open-source OCR engines.

In this section, we will mention few of the most popular ones, like *ABBYY FineReader* [ABBYY, 1993], *ReadIRIS* [IrisGroup] or *Omnipage* [Communications, 1980s]. We will look at their comparison report and mention a few of their features.

We will start this chapter off by looking more closely into **ABBYY FineReader.** This OCR engines claims to convert scanned PDF files into editable electronic formats like MS Word, MS Excel, RTF, HTML etc... It supports over 192 lan-

guages and even has a built-in spell check for 48 of them and includes the support for table and spreadsheet recognition, as well as batch processing of multiple documents. It also supplies SDKs for embedded and mobile devices. These are all the reasons why, to this date, it has over 20 million users. However, as described by Heliński et al. [2012] in a comparison report between FineReader and Tesseract, this engine has also its downsides. In some cases, Tesseract seemed to perform significantly better than ABBYY FineReader. This was mostly in the cases of Gothic fonts and good quality images, which FineReader had trouble processing. Moreover, FineReader seemed to have a problem with pages with great amount of small characters, as well as pages with small about of big characters, although the engine was trained to recognize them. However, when dealing with either noise or complicated segmentation, Tesseract mostly failed, while FineReader performed quite nicely.

Another popular software to look into is **ReadIris**. It provides similar features to ABBYY FineReader - batch processing of documents, conversion to MS Word, MS Excel or MS Powerpoint, splitting or merging PDFs and Calc table recognition. It also includes features like voice annotations. Its main difference from ABBYY FineReader is that it is less robust - it is not available for Linux, has only 138 languages compared to FineReader's 192 and the number of output formats is undoubtedly smaller. Although this might be a downside for user experience, the performance of ReadIris can sometimes even beat the one of FineReader. For example, in a study done by Safonov et al. [2011] on card recognition, ReadIris performed better than FineReader in both numeral and orientation detection. ReadIris is also widely used for the recognition of Arabic and Hebrew characters, where it performs even better than ABBYY FineReader.

Last, but not least, we are going to have a look at **Omnipage**. It promises its customer the same things that both already mentioned softwares - conversion of scanned documents into editable documents with various formats. In comparison to ReadIris, it supports similar number of languages. Its upside is the availability for Linux and greater amount of supported formats. However, in some image cases, its performance is compromised. As mentioned by de Mello and Lins [1999], Omnipage has great trouble working with colored images. It firstly transforms them into greyscale and only then performs the recognition. It also has problems dealing with rotation over 12 degrees, which simple preprocessing algorithms in combination with Tesseract should not find to be a problem. On the other hand, Omnipage provides a user-friendly environment and is optimized for speed, which contributes to the overall user experience.

These are by far not the only existing engines. When choosing a commercial OCR software, numerous other options promising similar features and results arise - like *SimpleOCR*, *Orpalis*, *Adobe Acrobat Pro DC* or *CVision OCR Engine*. However, these programs have almost no value for developers (except for comparison studies), whose work is still either concentrated on improving and expanding the Tesseract engine, or based on Tesseract's robust library.

// mention somewhere that .tif images are the best

2. Layout Recognition For Tabular Data

3. Table Recognition Implementation

- conversion of formats to .tiff format (tesseract doesn't know .pdf format) - preprocessing in the following steps - scaling, deskew, contrast, binarization, noise reduction, scanning border reduction, alpha channel?
 - // existuju aj ine transformacie, ako sharpening, alpha correction etc, add small border

4. Results

Conclusion

Bibliography

- ABBYY. Abbyy finereader. https://abbyy.technology/en:kb:images_resolution_size_ocr. Accessed: 2018-12-03.
- ABBYY. Abbyy finereader. <https://www.abbyy.com/en-ee/finereader/>, 1993. Accessed: 2018-12-05.
- Salem Saleh Al-amri, Dr. N. V. Kalyankar, and Dr. S. D. Khamitkar. Linear and non-linear contrast enhancement image mr. 2010.
- Wojciech Bieniecki, Szymon Grabowski, and Wojciech Rozenberg. Image preprocessing for improving ocr accuracy. In *Perspective Technologies and Methods in MEMS Design, 2007. MEMSTECH 2007. International Conference on*, pages 75–80. IEEE, 2007.
- Dan Bloomberg. Leptonica library. <http://www.leptonica.com/>, 2001. Accessed: 2018-12-04.
- Antoni Buades, Bartomeu Coll, and J-M Morel. A non-local algorithm for image denoising. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 2, pages 60–65. IEEE, 2005.
- M Ćadík. Perceptual evaluation of color-to-grayscale image conversions. In *Computer Graphics Forum*, volume 27, pages 1745–1754. Wiley Online Library, 2008.
- Arindam Chaudhuri, Krupa Mandaviya, Pratixa Badelia, Soumya K Ghosh, et al. *Optical character recognition systems for different languages with soft computing*. Springer, 2017.
- Nuance Communications. Omnipage. <https://www.nuance.com/print-capture-and-pdf-solutions/optical-character-recognition/omnipage/omnipage-ultimate.html>, 1980s. Accessed: 2018-12-05.
- John Cristy. Imagemagick. <https://www.imagemagick.org/script/index.php>, 1999. Accessed: 2018-12-04.
- Carlos AB de Mello and Rafael D Lins. A comparative study on ocr tools. In *Vision Interface*, volume 99, pages 224–231. Citeseer, 1999.
- Vikas J Dongre and Vijay H Mankar. A review of research on devnagari character recognition. *arXiv preprint arXiv:1101.2491*, 2011.
- R Dorothy, RM Joany, R Joseph Rathish, S Santhana Prabha, and S Rajendran. Image enhancement by histogram equalization. *Int. J. Nano. Corr. Sci. Engg*, 2(4):21–30, 2015.
- Shreyas Fadnavis. Image interpolation techniques in digital image processing : An overview. 2014.

- Dianyuan Han. Comparison of commonly used image interpolation methods. In *Proceedings of the 2nd International Conference on Computer Science and Electronics Engineering*. Atlantis Press, 2013. ISBN 978-90-78677-61-1. doi: 10.2991/iccsee.2013.391. URL <http://dx.doi.org/10.2991/iccsee.2013.391>.
- Marcin Heliński, Miłosz Kmiecik, and Tomasz Parkoła. Report on the comparison of tesseract and abbyy finereader ocr engines. 2012.
- Itseez Intel Corporation, Willow Garage. Opencv library. <https://opencv.org/about.html>, 2000. Accessed: 2018-12-04.
- IrisGroup. Readiris. <http://www.irislink.com>. Accessed: 2018-12-05.
- Christopher Kanan and Garrison W. Cottrell. Color-to-grayscale: Does the method matter in image recognition? *PLOS ONE*, 2012. doi: 10.1371/journal.pone.0029740. URL <https://doi.org/10.1371/journal.pone.0029740>.
- Yue Lu and Chew Lim Tan. A nearest-neighbor chain based approach to skew estimation in document images. *Pattern Recognition Letters*, 24(14):2315–2323, 2003.
- William B Lund, Douglas J Kennard, and Eric K Ringger. Combining multiple thresholding binarization values to improve ocr output. In *Document Recognition and Retrieval XX*, volume 8658, page 86580R. International Society for Optics and Photonics, 2013.
- Song Mao and Tapas Kanungo. Empirical performance evaluation methodology and its application to page segmentation algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(3):242–256, 2001.
- G Maragatham, S Md Mansoor Roomi, and P Vasuki. Noise detection in images using moments. *Research Journal of Applied Sciences, Engineering and Technology*, 10(3):307–314, 2015.
- Nadira Muda, Nik Kamariah Nik Ismail, SA Abu Bakar, and Jasni Mohamad Zain. Optical character recognition by using template matching (alphabet). In *National Conference on Software Engineering & Computer Systems*, 2007.
- C Mythili and V Kavitha. Efficient technique for color image noise reduction. *The research bulletin of Jordan, ACM*, 1(11):41–44, 2011.
- Hewlett Packard. Tesseract about. <https://github.com/tesseract-ocr/tesseract/wiki/ReadMe>. Accessed: 2018-12-04.
- Hewlett Packard. Tesseract improve quality. <https://github.com/tesseract-ocr/tesseract/wiki/ImproveQuality>, 1996.
- Worapoj Peerawit and Asanee Kawtrakul. Marginal noise removal from document images using edge density. In *4th Information and Computer Engineering Postgraduate Workshop, Phuket, Thailand*. Citeseer, 2004.

- Daniel Rosner, Costin-Anton Boiangiu, Mihai Zaharescu, and Ion Bucur. Image skew detection: A comprehensive study. In *Proceedings of IWoCPS-3, The Third International Workshop On Cyber Physical Systems, Bucharest, Romania*, 2014a.
- Daniel Rosner, Costin-Anton Boiangiu, Mihai Zaharescu, and Ion Bucur. Image skew detection: A comprehensive study. In *Proceedings of IWoCPS-3, The Third International Workshop On Cyber Physical Systems, Bucharest, Romania*, 2014b.
- I Safonov, H Lee, SH Kim, and D Choi. Intellectual two-sided card copy. In *Proc. of GRAPHICON-2011*, pages 38–41, 2011.
- Jaakko Sauvola and Matti Pietikäinen. Adaptive document image binarization. *Pattern recognition*, 33(2):225–236, 2000.
- Mehmet Sezgin and Bülent Sankur. Survey over image thresholding techniques and quantitative performance evaluation. *J. Electronic Imaging*, 13:146–168, 2004.
- Faisal Shafait and Thomas M. Breuel. A simple and effective approach for border noise removal from document images. *2009 IEEE 13th International Multitopic Conference*, pages 1–5, 2009.
- Faisal Shafait, Daniel Keysers, and Thomas Breuel. Performance evaluation and benchmarking of six-page segmentation algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(6):941–954, 2008.
- Bhavesh Kumar Shukla, Gautam Kumar, and Ashish Kumar. An approach for skew detection using hough transform. *International Journal of Computer Applications*, 136(9):20–23, 2016.
- Ruby Singh and Ramandeep Kaur. Improved skew detection and correction approach using discrete fourier algorithm. *International Journal of soft computing and Engineering*, 3(4):5–7, 2013.
- P Suganya, S Gayathri, N Mohanapriya, et al. Survey on image enhancement techniques. *International Journal of Computer Applications Technology and Research*, 2(5):623–meta, 2013.
- Oivind Due Trier, Anil K Jain, Torfinn Taxt, et al. Feature extraction methods for character recognition-a survey. *Pattern recognition*, 29(4):641–662, 1996.
- Jamileh Yousefi. Image binarization using otsu thresholding algorithm. 2011.

List of Figures

List of Tables

List of Abbreviations

A. Attachments

A.1 First Attachment