



**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 Deskew	6
1.1.3 Contrast enhancement	7
1.1.4 Binarization	9
1.1.5 Noise Reduction	10
1.1.6 Scanning border Reduction	12
1.2 Basic Techniques	12
1.3 Available Implementations	13
2 Layout Recognition For Tabular Data	14
3 Table Recognition Implementation	15
4 Results	16
Conclusion	17
Bibliography	18
List of Figures	19
List of Tables	20
List of Abbreviations	21
A Attachments	22
A.1 First Attachment	22

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 are a lot of factors that complicate this process. Based on different OCR documentations(...zdroje?), the few of the most important problems were found to be: To name a few of the most important:

- **different fonts** - Nowadays, there is a number of fonts and styles. ?? dopisat 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's 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, however, the characters that are supposed to be the same don't always have to look that way, (because of people/given the mistakes of people). This is why the OCR software has to count on (???) minor mistakes the odchylka depends 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're 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.

- ... noise?

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

OCR algorithms usually don't work well on most images provided by user. It's because they(???) are either malformed, disrupted or have other of the problems mentioned above.

Most of the OCR engines already come with some kind of built-in preprocessing filter. The problem with those built-in filters is that they most likely don't match your 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 result of the OCR.

// existuju aj ine transformacie, ako sharpening, alpha correction etc, add small border

1.1.1 Scaling

Usually in case of OCR, more is better. The more information the OCR engine can use to interpret text, the more accurate the result will be. This can be specifically applied to the case of resolution and bit-depth.

(zdroje?) Almost all OCR engines(Tesseract, OpenCV, ...zdroje?) encourage their users to use a 300 DPI images. Their reason for that is pretty simple - it's the point where you gain the most accuracy without sacrificing speed and file size. For example, try running a few tests for the same image with different resolution on OCR engines. (after running a few tests? zdroje?) You'll see that the improvement gap between 200 DPI scan and 300 DPI scan will be at least 2 times the improvement gap of any other resolutions. Also, comparing other images above 300 DPI with a 300 DPI image, you find the improvement gap to be nearly absent. It's 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 (to do - ktore?), 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 (unsharp???). The most

affected parts would be the diagonal edges of elements. They would appear jagged (*aliased*). (add a sample picture somewhere over here !!!). (This 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.

(adaptive https://www.researchgate.net/publication/224647180_Adaptive_Interpolation_Along_Time_Image_Resizing)

Non-adaptive methods don't 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.

- **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.

(-i https://www.researchgate.net/publication/301889708_Image_Interpolation_Techniques_in_Digital_Image_Processing)

There also exist higher order interpolations, such as *Spline*, *Sinc*, *Lanczos*... They take more surrounding pixels into consideration and calculate the resulting value through (more complicated functions? to-do - pochopit ako to presne robia, netreba ak dam referenciu?). The results are better than just simple calculation, but they take up way more time. During OCR, such complex and time-consuming functions are not necessary, as we aren't 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 (zase spomenut tesseract, opencv etc?) is *Bicubic Interpolation*. Although *Nearest Neighbor* and *Bilinear* methods are extremely fast, the results were found to be poor.

The more pixels, the more accurate the interpolation result. This obviously comes at the expense of a processing time.

- **DPI of new image > 300 :** Although we are solving the exact opposite problem as in previous case, the approaches to this are quite similar. In this case, though, we need to 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 easiest 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 (as in previous case, the 4x4 turns out to be working ... best). We calculate their weighted average and assign this value to the unknown pixel.

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 Deskew

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

One of the steps of OCR algorithms is *page segmentation*. It's the process that decomposes a document image into elements. This division works with (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, what we need 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 after binarization of the image (for more accurate results) and generally assume the skew angle to be no more than 15°. As mentioned in (ref), algorithms that work on greater skew angles also exist. 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.

To name a few of the most promising algorithms:

- **Hough Transform:** As described in (hugh ref), this method is used to find straight lines in image and given their rotation, determine the skew angle of the whole documents. It works by saving results of Hough Transform on different lines, and comparing which line has the most pixels. That gives us the line that is most represented in the input image, which gives us the resulting angle. Although highly effective, computations of transform

can be quite time consuming. This is why Hugh Transform usually isn't done on every point of the image, but only a set of chosen input points. Another problem occurs if pictures are present in the input image (as Hugh Transform counts with aligned pixels). To solve this, we can't choose input points from the given picture.

- **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 (ccref), 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. (maybe drawbacks? As the algorithms greatly differ, each has its own drawbacks. cc vzdý rovnaky algo)

Worth noting are also other methods like *Fourier*, *Wavelet Decomposition*, *Radon Transform*, *PCP*, *one-pass* or *multi-pass* skew correction, *morphology algorithms* etc... These are all briefly described in (ref).

(talk more about fourier?)

- fourier - <http://www.ijscce.org/wp-content/uploads/papers/v3i4/D1739093413.pdf>

- ccref - <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.121.3963rep=rep1type=pdf>

- hugh ref - <https://pdfs.semanticscholar.org/a995/14ebd9caf97ec8fa80da58bcb0f9b2fb93e1.pdf>

- one pass skew - https://ac.els-cdn.com/S1319157803800034/1-s2.0-S1319157803800034-main.pdf?_id=7f3311c1-8a8f-48be-888d-7a320f491dce&cdnat=1543157733_f26c35ff33referencia-https://www.researchgate.net/publication/263540666_image_skew_detection_ACOR

1.1.3 Contrast enhancement

A *histogram* is an accurate representation of the distribution of data. In case of image histograms, it represents the tonal distribution of the image - x-axis in histogram stands for all available tonal levels, and y-axis represents the number of pixels for each tonal level. (mby picture?)

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's 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 simply find the average of the R, G and B values. This does not really preserve the brightness of the image. Therefore, corrections like luma 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 in (zdroj other grayscale more), 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. Following are a few methods that approach this problem. Other types of methods can be found (refs)

- **Linear Stretching** method linearly expands the original digital values of the histogram into a new distribution. As mentioned in (ref), there exist three methods of linear stretching - *Min-Max*, *Percentage* and *Piecewise Contrast Stretch*. 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* mentioned in (ref). 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. As described in (ref), this method is based on transforms dependant on the probability distribution function of the histogram. Although

it produces unrealistic effects in photographs, it's widely used for scientific images.

- logarithmic and power law - <http://academica-e.unavarra.es/bitstream/handle/2454/2368/>
- linear stretch: http://www.academia.edu/9301934/Comparative_study_of_linear_and_non-linear_contrast_enhancement_techniques-histogram_eq : https://www.researchgate.net/publication/260111111_linear_contrast_enhancement_techniques-histogram_eq
simple grayscale : <https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0020784>
other grayscale more : http://cadik.posvete.cz/color_image_valuation/cadik08_perceptual_evaluation_of_linear_and_gamma_contrast_enhancement_techniques
other grayscale : <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.221.4952&rep=rep1&type=pdf> - <https://arxiv.org/ftp/arxiv/papers/1003/1003.4053.pdf> - <https://www.sciencepubco.com/index.php/ijet/article/view/14811/6062>

1.1.4 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, as mentioned in ref (zdroj).

This is when binarization comes in handy. 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.

As mentioned in (zdroj global), the simplest approach to binarization is *global thresholding*. It works on choosing a threshold value and iterating over the whole image, comparing the value of each pixel to this threshold. If it's greater, the pixel in resulting image will be black, otherwise white. As the brightness and contrast of different images vary, it's impossible to choose a threshold suitable for all images.

Therefore, other more complex methods are used:

- **Otsu's method** 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's simple and easy to implement. However, upon applying it to images with uneven dark or bright spots, it (shows/deems to be?) unusable.
- **Local Thresholding:** 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 (somewhere here reference thresholding?). 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 other. 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. Their approach is to analyze the document image surface and determine the most useful method. (hybrid switching? asi sa zamotam). More information on this subject can be found (ref adaptive) (for more information on?i)

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 (ref), (ref) or even in the articles already mentioned in this section.

- further possibilities - http://www.busim.ee.boun.edu.tr/sankur/SankurFolder/Threshold_
- adaptive - <http://www.mediateam.oulu.fi/publications/pdf/24.p>
- otsu - https://www.researchgate.net/publication/277076039_Image_Binarization_using_Otsu
- global thresholding https://www.researchgate.net/publication/236029409_Combining_Multi
- thresholding - <https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7784945>
- ref - why binarize https://www.researchgate.net/publication/287206352_Prediction_of_the_Opti

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. Edge detection is based on the differences between the "outside" of the element and its "inside". When noise is present, it can disrupt these edges and the exact (boundary?border?) is therefore changed and hard to detect.

There exist different type of noise:

- **salt and pepper noise** - also known as impulse noise, it 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.

Given the type of noise, different approaches are used for its reduction. To name a few of the most popular (zdroje?):

- **linear filters** - like *mean filter*, *Gaussian filter* or *Wiener filter*. As described in (zdroje?), these filters are usually simple and easy to implement, fast and therefore widely used. 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.
- **non-linear filters** - (zdroje o median filtry) the most popular 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.
- **non-local means method** - (mentioned in ..zdroj... ocr article) takes a mean value of all pixels, weighted by how similar these pixels are to the target pixel. This method produces an even sharper and clearer image results as mentioned filters, zase spomenut cas

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's 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 (examples, zdroje), 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. 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. (ako presne funguju -¿ zdroje)

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 (or *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 either done by a *black filter* (a moving rectangular window that calculates the ratio of black pixels on border zdroj), a sequential algorithm, a two-pass algorithm... 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* - rovnaky zdroj ako black) - 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, margings 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 in (zdroj edge detector). It's 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 - zdroj) 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.

edge detector - <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.60.3969rep=rep1>

black filter - https://www.researchgate.net/publication/224100168_Asymptotically_efficient_black_filter

<https://www.rug.nl/research/portal/files/14632744/thesis.pdf>

1.2 Basic Techniques

Nowadays, two approaches are proposed for OCR. The first one is an approach based on heuristics -

- HMM approach (hidden markov)
- neural networks

1.3 Available Implementations

// mention programs for preprocessing // tesseract, opencv etc
// mention somewhere that .tif images are the best

2. Layout Recognition For Tabular Data

3. Table Recognition Implementation

4. Results

Conclusion

Bibliography

List of Figures

List of Tables

List of Abbreviations

A. Attachments

A.1 First Attachment