Enhancing Images utilizing advanced Binarization Techniques

Algorithm Engineering 2023 Project Paper

Tarek Al Mustafa
Friedrich Schiller University Jena
Heinz Nixdorf Chair for Distributed Information Systems
Germany
tarek.almustafa@uni-jena.de

ABSTRACT

KEYWORDS

image binarization, adaptive thresholding, algorithm engineering

1 INTRODUCTION

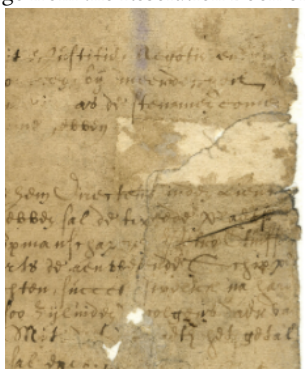
Image binarization has been an active field of research for over 40 years[6]. It is one of the premier methods to enhance and clean images of texts, and therefore has great importance for applications such as machine learning, or the preservation of historical documents[8, 9]. However, even after decades of research, image binarization is still an open problem, even being included in ICDAR[1], a competition for document analysis and recognition[5]. In this paper, we implement an existing approach to solving image binarization using local minima and maxima[7] and document our findings.

2 BACKGROUND AND RELATED WORK

The main task of image binarization is to convert a coloured or gray-scale input image of written text into an output picture that contains only black or white pixels. In the best case, all background pixels end up white, and all text pixels become black. Therefore, there would be a vast contrast between the text and the background. For machine learning applications, this makes it easier to classify pixels and consequently process printed or hand-written text. Image binarization also supports the preservation of historical documents. These documents however, highlight the challenges of this task: Different lightning conditions, camera qualities, or degraded paper introduce great difficulty. This can be seen in Figure 1. Next to the considerable challenges from image inputs, there are several others stemming from the implementation and computational load. It is imperative that an implementation solving this task is optimized for two main reasons: (1) pre-processing takes into account every pixel of the input image, meaning that complex operations with high execution time slow down the process exponentially and (2) when used for machine learning applications, every input image of at least multiple tens of thousands must be pre-processed. This introduces another layer of complexity. Therefore the main task is not only to find the best solution for image binarization, but to also balance performance.

Thresholding. The simplest approach to binarization is to define a threshold and compare every pixel value to it. If the pixel is over the threshold, it becomes white, and if not, it becomes black.

a) First page from the Resolution Book of Curaçao.[a]



[a]https://www.newnetherlandinstitute.org/research/online-publications/new-york-historical-manuscripts-dutch-new-netherland-documents-series-introductions/

b) Picture of Textbook with difficult lighting.

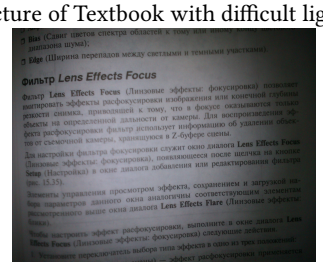
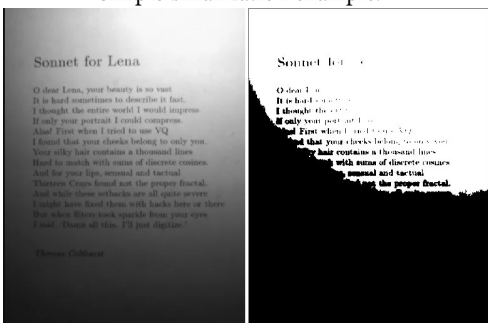


Figure 1: Examples of challenging image inputs.

Simple binarization example.[a]



[a]https://towardsdatascience.com/pre-processing-in-ocr-fc231c6035a7

Figure 2: The limitations of simple thresholding.

The biggest limitation of this approach is that it fails as soon as slightest lighting changes are introduced as seen in Figure 2. Thresh-

[1]https://icdar2021.org/program-2/competitions/

olding can be improved by computing the average pixel value of the whole image, and deriving the threshold value from that average, however, the same lighting problem persists.

Otsu's Method. Otsu's Method[4] is a thresholding technique that searches for the threshold, that maximizes the between class variance of two classes; meaning that it separates the foreground from the background. This is done by (1) computing the gray-value histogram of the whole image, (2) setting a threshold based on that histogram and defining foreground and background from the threshold, (3) computing the weights for back- and foreground, (4) gaining the average class intensity from those weights, and (5) computing the between class variance from the average intensities. By computing this variance for multiple threshold candidates, the candidate with the highest between class variance is selected for the whole image. This method can be used for multiple applications including separating text from its background.

Adaptive Thresholding. Since global thresholds fail to solve this task, local thresholds can be used to successfully combat lighting changes[1–3]. Adaptive thresholding techniques find various ways to compute a threshold dynamically, based on properties of the pixels that surround the one under consideration. The main issue with varying lighting is that adaptive thresholding techniques usually work well on dark images, but they perform considerably worse on very light inputs, meaning that in the input image, not only the background is very white, but also the text on the image appears light-gray to white because the original lighting was very bright. For this work, we chose to implement an adaptive thresholding method that can solve both background and lighting tasks[7].

3 THE ALGORITHM

We implement an adaptive thresholding technique that dynamically computes local thresholds around candidate pixels. Our implementation uses no outside dependencies or libraries besides *stb*[2] to read and write input and output image files. This supports better portability and opens more windows for optimization. We enhance performance using *OpenMP*[3]. We describe out implementation below:

STEP 1 : We use the *stb* library to read image files. These files can have varying resolutions or colour values. Therefore, the first step is to convert possible RGB images into gray-scale images. We solve this issue by iterating over every pixel from the input image, summing up the respective red, blue, and green values and dividing the sum by 3. If the input image has a 4th alpha channel, we take the alpha value into account by multiplying it with a colour channel. Since the iteration is a for-loop, we optimize performance by using *OpenMP's pragma omp parallel* keyword that automatically improves runtime. Through this step, we transform an input image into gray-scale. *STEP* 2 : Our goal is to perform binarization on gray-scale input images. In our approach, we want to take locality into account to better combat the issues of different lighting effects or varying backgrounds. The intuitive approach is as follows:

We determine whether the current candidate pixel is either a high contrast pixel, or a normal pixel. Intuitively, the candidate pixel has a high likelihood of being in a bright region of the image, if a lot of surrounding pixels in its neighbourhood have high gray-scale values as well. We determine this by computing the average gray-scale value of the surrounding pixels, and then counting how many of those neighbours have gray-scale values above that average. To that end, we introduce two important parameters - the window size (meaning the size of the area that surrounds our candidate pixel; an odd integer > 1), and the bound of bright pixels (the number of neighbours whose values are above the window's average). These two parameters determine the accuracy of our solution and should be tuned for different inputs.

Low brightness candidates. If our candidate pixel is determined to not be in a bright region, we determine the maximum and minimum gray-scale values in the candidate window and compute the local threshold of that window with using: $(Max-Min)/(Max+Min+c)$, c being a very small constant to avoid division by 0. If the candidate pixel is below this threshold, it is coloured black. If not, it is coloured white.

High brightness candidates. For these candidates, we check a new condition. When this condition is satisfied, the candidate is coloured white. We first compute the mean and standard deviation of the image intensity around the candidate and then check whether the candidates intensity is $<= Mean + StDev/2$. If the candidate does not satisfy this condition, it is coloured black.

Once these steps are performed on every pixel of the input image, the produced output image consists of only black and white pixels.

Parameters. As stated above there are two tunable parameters for this algorithm, the window size and neighbourhood bound. As explained in[7], both of them are "correlated to the width of the text strokes within the document image under study". The window size should not be smaller than the size of the text within the image, otherwise the inside of the letters within the text might be classified incorrectly. For example, if an "I" in an input image has a width of 7 pixels, the window size is set to 3, and we iterate over the letter from left to right; then we lack the context for when the letter starts on the left side, and when it ends on the right side. If that is in a high brightness region, then there might not be enough bright pixels to satisfy the neighbourhood bound condition (because we lack the bright pixels from the right end of the letter). Therefore, the candidate might be coloured incorrectly. As for the neighbourhood bound itself, should be set such that both left and right edges of a letter can be taken into account.

4 EVALUATION AND DISCUSSION

4.1 Example Images

We evaluate our approach on example use cases, shown in Figures 3, 4, and 5 below. The input image of figure 3 is a good example of a possible worst case. The text is blurred in some areas, and the lighting varies from being almost completely dark at the edges to being so bright in the centre, that it changes the text colour. In the output image, we can see that the dark edges have been removed completely and the text is generally legible, however our approach struggles to identify clean edges in the bright centre area. While the centre might still be readable for a human, a computer would not be able to identify the letters with high accuracy. The second example shows an input close to a best case scenario, the picture has high resolution and no significant bright spots. As a result,

[2]https://github.com/nothings/stb
[3]https://www.openmp.org/

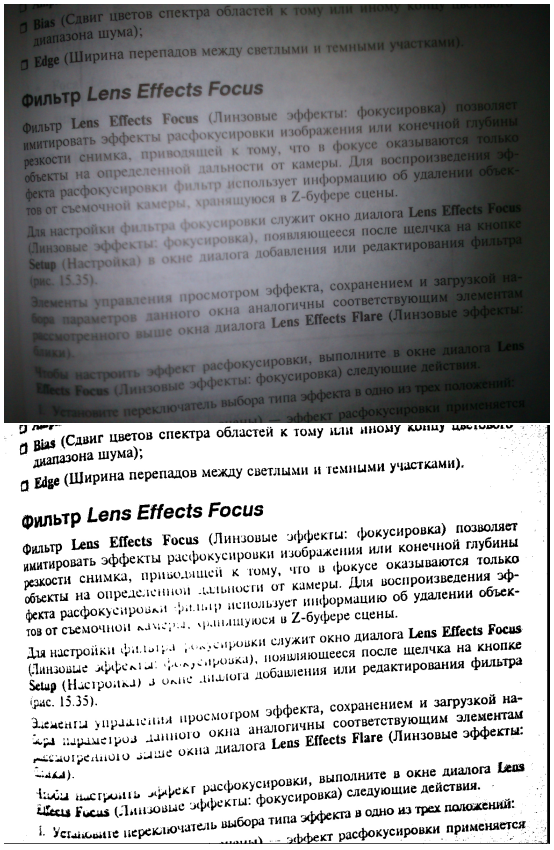


Figure 3: Binarization example 1: Window size = 3x3, Bright pixel bound = 20

the output image produces perfectly legible text and almost the entire background is white. The third image test the background separation capabilities of the algorithm. It has multiple dark areas that surround the pink metro lines and blue rivers. In the output, we can see that the background has been removed with high accuracy and the handwritten and printed text is legible, however some of the pink lines were classified as white pixels. This means, that lighter colours such as pink or yellow might be miss-classified more often than darker colours.

From these example images we can draw the following insights about the accuracy of our approach:

- The best case scenario inputs are processed with high accuracy.
- On input images with very bright spots, our approach is error prone. Dark spots in images are processed better than bright spots.
- Our approach performs better on images with a high resolution, than on images with a low resolution since lower resolution images generally have blurrier text.
- Impurities in the background are processed with decent accuracy.
- Bright colours in the input image can introduce errors to the output.

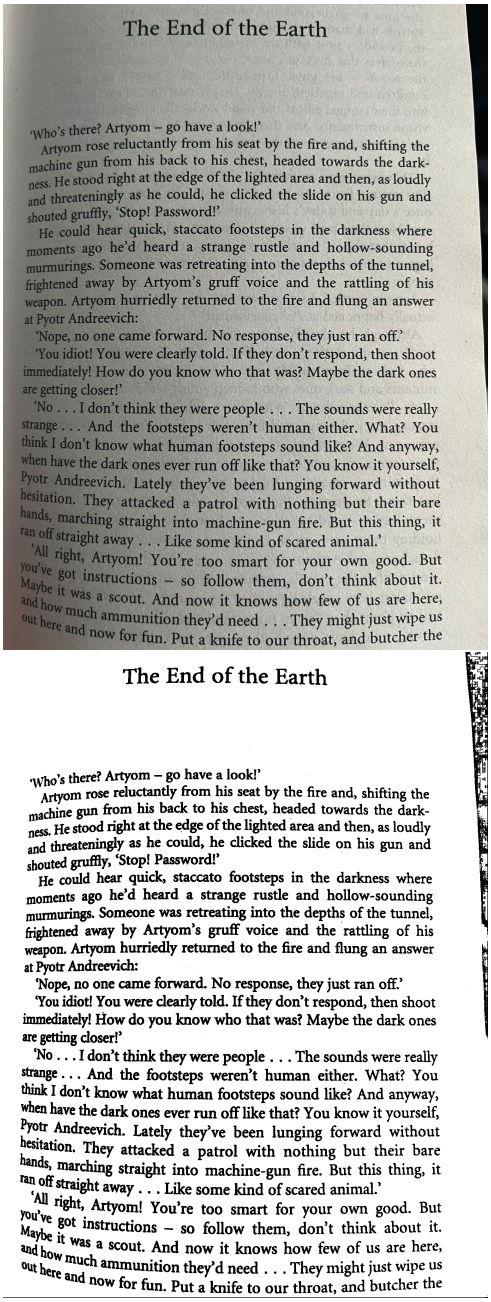


Figure 4: Binarization example 2: Window size = 3x3, Bright pixel bound = 6, Book = Metro 2033 :)

4.2 Performance Optimization

Our implementation leaves room for further optimization, we will list some possibilities here:

- Parallelizing the binarization process would lead to a significant increase in performance. This however, would have to be done by hand since the algorithm is too complex to be handled by a simple *pragma omp parallel* expression; in

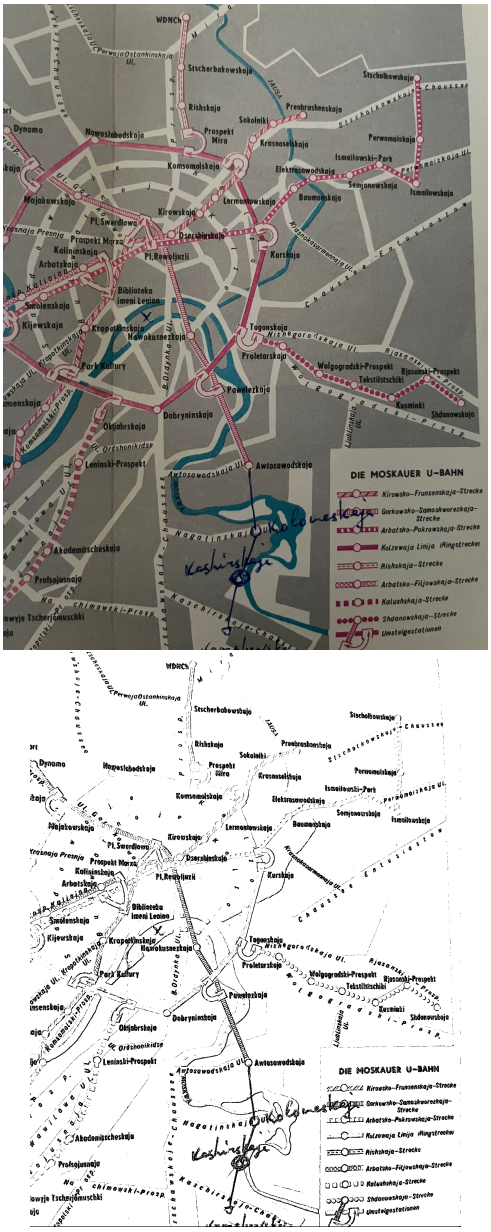


Figure 5: Binarization example 3: Window size = 9x9, Bright pixel bound = 4

fact, using the same expression on our current code, makes performance worse by a factor of 5.

- Current-window-averages and local minima/maxima could be stored/cached such that for every new pixel, only a few new operations have to be executed. Currently, for every new candidate pixel, all operations are executed from scratch.
- Since image pixels and their windows are independent from far away pixels, many threads can be utilised to work on different quadrants of the input image.
- Every time the candidate pixel is in a bright area, the program could spawn a new thread, such that the next pixel can already be pipelined. In the worst case this would create a queue of pixels waiting to be processed (the same as our current program), but in the best case a new candidate could already be processed while an old one is still being worked on.

4.3 Discussion

Our implemented binarization algorithm works well on a variety of different input images. Even though it introduces errors in some scenarios, a so far not discussed upside of this approach is that the algorithm is simple to implement, optimize, and already has decent performance. Even though this papers main topic is binarization, we must not forget that binarization is only one of many possible pre-processing steps to prepare inputs for machine learning tasks. Other keywords and processes include, but are not limited to: Background removal, skew correction, noise removal, thinning, skeletonization, segmentation, and many more. Therefore, we conclude, that the proposed method, in combination with multiple other methods, solves the task of binarization well.

5 CONCLUSION

In this paper we introduced an approach to solving the image binarization task as part of the machine learning pipeline or part of the preservation of text documents. The algorithm is based on computing local minima and maxima in windows of various sizes around candidate pixels, and also proposes an approach of processing pixels in high brightness areas based on neighbouring mean and standard deviation statistics. We evaluated our implementation on a variety of example images, discussed its strengths and shortcomings, and proposed multiple methods with which performance could be optimized further. Future work includes performance optimization, inclusion of more pre-processing steps after binarization - especially background removal and skeletization - more comprehensive evaluation on real datasets, and formalized evaluation of how different window sizes and bright neighbour bounds impact performance.

REFERENCES

[1] Jérôme Bobin, Jean-Luc Starck, Jalal M Fadili, Yassir Moudden, and David L Donoho. 2007. Morphological component analysis: An adaptive thresholding strategy. *IEEE transactions on image processing* 16, 11 (2007), 2675–2681.
[2] Derek Bradley and Gerhard Roth. 2007. Adaptive thresholding using the integral image. *Journal of graphics tools* 12, 2 (2007), 13–21.
[3] Francis HY Chan, Francis K Lam, and Hui Zhu. 1998. Adaptive thresholding by variational method. *IEEE Transactions on Image Processing* 7, 3 (1998), 468–473.
[4] Takio Kurita, Nobuyuki Otsu, and N Abdelmalek. 1992. Maximum likelihood thresholding based on population mixture models. *Pattern recognition* 25, 10 (1992), 1231–1240.
[5] Rafael Dueire Lins, Rodrigo Barros Bernardino, Elisa Barney Smith, and Ergina Kavallieratou. 2021. ICDAR 2021 competition on time-quality document image binarization. In *Document Analysis and Recognition–ICDAR 2021: 16th International Conference, Lausanne, Switzerland, September 5–10, 2021, Proceedings, Part IV 16*. Springer, 708–722.
[6] Paul W Palumbo, Puducode Swaminathan, and Sargur N Srihari. 1986. Document image binarization: Evaluation of algorithms. In *Applications of Digital Image Processing IX*, Vol. 697. SPIE, 278–285.
[7] Bolan Su, Shijian Lu, and Chew Lim Tan. 2010. Binarization of historical document images using the local maximum and minimum. In *Proceedings of the 9th IAPR International Workshop on Document Analysis Systems*. 159–166.

[8] Chris Tensmeyer and Tony Martinez. 2020. Historical document image binarization: A review. *SN Computer Science* 1, 3 (2020), 173.

[9] Wei Xiong, Xiuhong Jia, Dichun Yang, Meihui Ai, Lirong Li, and Song Wang. 2021. DP-LinkNet: A convolutional network for historical document image binarization. *KSII Transactions on Internet and Information Systems (TIIS)* 15, 5 (2021), 1778–1797.