AN EFFICIENT METHOD FOR A SPECIFIC CASE OF DETECTING IMPULSE NOISE ON SCANNED DOCUMENTS

Petar Prvulović[1], Jelena Vasiljević[1], Dhinaharan Nagamalai[2]

[1]School of Computing, Union University, Belgrade, Serbia
[2]Department of Computer Science and Engineering, Kalasalingam Academy of Research and Education, Krishnankoil-626126, India

ABSTRACT

This paper explains a method used to detect the presence of impulse noise in a set of scanned documents as a part of OCR preprocessing. As the document set is supposed to be processed in large scale, the primary concern of the noise detection method was efficiency within existing project constraints. Following the nature of noise, the method seeks to detect the presence of noise in document margins. The method works in two stages. First stage is margin detection, based on color spectre analysis. Second stage is noise recognition in margin samples, based on a pixel contrast score. The resulting implementation proved efficient both in terms of detection accuracy and algorithmic complexity.

KEYWORDS

Impulse noise, Noise detection, Margin detection

1. INTRODUCTION

The problem we faced and developed a method to solve is the detection of specific cases of impulse noise in a set of scanned documents. Solution needed to take in account the boundaries of the project which introduced a method fully adapted to the document set and underlying code base. Document set features a large number of business reports, in the order of millions, scanned as PDF files where each page is a separate JPEG file. Pages were scanned in grayscale CCITTFax format. Data processing program, which the method proposed here is part of, is implemented in Python 3 and uses numpy and OpenCV for other processing steps. These constraints do not affect the essence of the method as the method itself doesn't use any external functions, but they do dictate the coding style [1].

By inspecting the documents manually, several types of noise were noticed [2, 3]. Impulse noise is uniformly distributed over the document. Stains are mostly present in corners of pages as they originate from stapling the paper. Lines, both horizontal and vertical, appear near the edges. Impulse noise presents the largest problem and is in the focus of the presented method. We have taken into account the other two types of noise, so as to make the presented method resistant to their appearance, as it was noticed that it affects noise detection, as explained later in this paper.

When impulse noise is present, the OCR process introduces errors. A variety of despecling filters is available and some of them proved good in noise removal in this specific case. In a manually processed sample it was noticed that applying a despeckling filter on noisy documents improves

David C. Wyld et al. (Eds): CMLA, CIoT, CSEIT, NCS, NLPD, SPM - 2021
pp. 175-187, 2021. CS & IT - CSCP 2021 DOI: 10.5121/csit.2021.111514

OCR accuracy, but increases error rate if applied on a clean document. Hence it is necessary to accurately check for noise presence, in order to apply a despeckling filter only where needed.

Our aim was to design a method which reduces both the number of passes and the coverage of each pass as a strategy to save on execution time and complexity. Nature of impulse noise in the dataset allows for noise detection in areas without text, where page margins are a good candidate for sampling. On a uniform background it is possible to detect noise by analysing the color spectre of pixels. Both steps, margin detection and noise detection by color spectre, can be implemented in an efficient manner and within given constraints.

1.1. Related Work

There is a variety of noise detection techniques, as noted in [4,5,6]. Most common way of noise detection is to process the document with a noise removal filter and measure the difference of original vs. the filtered version. If there is no significant difference the document is marked as noise-free, because noise removal didn't introduce any changes. Drawback of this approach is that there are several passes over the entire document: to apply noise removal filters, to measure the difference, and more, depending on the nature of the detection process.

Document margin detection, in our specific case, is a special case of a more general text detection problem. As neural networks gained popularity, recent research seem to be focused more on general cases like paper and text detection and recognition in videos and images in a natural scene. There is a number of approaches to this problem, which provide decent results. Some of them can be found in [7, 8]. In [7] authors detect four types of text images: document images, scene images, born-digital images and heterogeneous text images. Our case falls into first type. Methods mentioned include: edge extration and smoothing, clustering pixels with similar color and other methods aimed towards detecting the text region; detecting discriminative features of characters in text; segmenting the image and detecting the blocks which contain text; etc. As the purpose of these methods is to define a boundary in which the text is contained, they are designed in such a way that they need to pass through the entire image, which is the characteristic we wanted to avoid in our specific case.

1.2. Selected Approach

As previously noted, our aim was to detect page margin area and use it to sample the image and detect if noise is present using the color spectre. Having a very specific case of scanned documents, methods that rely on text boundary detection introduce unneeded processing steps as they need to pass through the entire image. We avoided that, and reduced the page processing to 40% of the page area by using the fact that the pages in our set of scanned documents are scanned straight, without rotation, and that the paper covers the entire image, hence eliminating the need to detect the page text boundary.

The proposed method works in two stages. First stage detects document margins using color spectre. Second stage checks if impulse noise is present within margins, as we treat them as an empty area on a document where noise is easily distinguished. Method is implemented as a main function and a set of helper functions. Main function returns boolean, indicating that the provided bitmap contains noise.

This paper is organised as follows: we first explain prerequisites and define an efficiency boundary which should be met in order to improve execution time of the entire process; then we develop both stages of the method. We conclude the paper with test results over a manually

prepared dataset where documents were separated into two groups: with and without noise. Test results prove that the efficiency of the proposed method is within acceptable boundaries.

2. PREREQUISITES

Manual inspection of the dataset showed that about 3% of documents contain impulse noise. These documents have to be pre-processed using a filter. We aim to save on execution time in the other 97% of documents by introducing a method which is faster than the noise removal process. Let's denote execution time of noise removal process as R and noise detection process as D. In case of using only noise removal process, overall execution time would be $T_1=1*R$. In case of using a detection process, noise removal would be performed only on 3% of samples. Overall execution time in this case is: $T_2 = 1*D + 0.03*R$. To achieve $T_2<T_1$, we need $D < 0.97*R$. This boundary is used in 6.1. Noise Detection Complexity, to confirm that the proposed method is within acceptable range.

Documents are scanned in portrait orientation, with little to no rotation. Where present, rotation is $< 0.5^{o}$. Page size varies around 3500px in height. Expected displacement of the upper and lower corner of the text area is set to be less than 16px. Each page is scanned as a JPEG picture in RGB format, though they are all in greyscale. Initial unpacking and transformation from PDF file gives us an nxm matrix of byte values, where each value represents a single pixel. Values range from 0-255, 0 being black and 255 being white. Documents seem to be preprocessed during the scanning, as all of them featured black text, there were no greyish and washed-out samples. Documents contain left aligned or justified text and/or tables, with clearly distinguishable margins. Nature of contents dictates uniform formatting. Typical page in the observed dataset is shown in Figure 1. Few cases were found with contents in header area, which doesn't affect our method as these sections are ignored, so cutting off a part of header or footer, with page number or similar, is acceptable.

Tables are often bordered and OCR showed better results when borders are removed, hence horizontal and vertical line removal is a required step and as such it is not considered as overhead if performed as part of noise detection method. This proved beneficiary in the margin detection stage. Line filtering is performed initially, after the page is loaded into memory as an OpenCV image. We applied a special horizontal and vertical kernel to detect lines and fill them with white pixels [9]. As this is a required pre-processing step for OCR, we do not count it into the noise detection process when we calculate the complexity.

Figure 1. Page example

3. MARGIN DETECTION

The method implements detection of left and right margins. We consider these two sufficient for noise detection in second stage. We expect that the outer side of the margin has significantly more light-colored pixels than the inner side. Margin color spectre is created by counting number of pixels for each of 256 levels of gray in bitmap columns. An example of spectre plot is depicted in Figure 2. and it shows the spectre of the first left quarter of a page depicted in Figure 1. Plot is generated using Python's matplotlib library. Black color indicates 0 values while white color indicates maximum value. Plot is in grayscale representation, which means that spectre values are scaled to [0-255] range. As there is disproportionately more white pixels, white color biases the plot, so to get better plot resolution we put a zero value for white. On Figure 2 we notice a sudden jump around column 350, which corresponds to left margin. x-axis indicates grayscale values (0-black, 255-white). y-axis indicates bitmap column.

As the left and right margins are vertically oriented and taking in account that rotation of text is less than 0.5º we expect that the transition step in color spectre is less than 16px wide, as stated in 2. Prerequisites. In Figure 2. we indeed notice a sharp jump in the darker part of the spectre, on the left side of the plot. Similar follows in the lighter part of the spectre. Gray shades are induced by interpolation and JPEG's lossy nature. Though this confirms the statement it is the consequence and not the reason and we will focus solely on the darker part of the spectre. With this, we reduced the problem of finding a margin to the problem of detecting a step in the spectre plot.

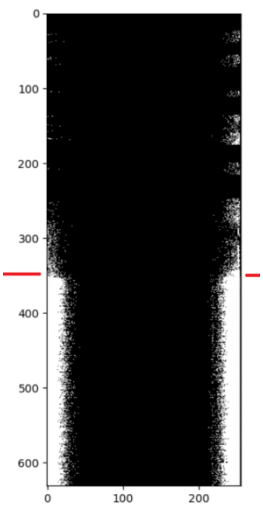

Figure 2. Page color spectre plot

Figure 3. Reduced spectre

We see no difference in black and “almost black” pixels, as they both represent a symbol on a page. Spectre color range is reduced to 8 groups, so the first group represents shades from 0-31, second from 32-63 etc. Figure 3. shows the reduced spectre, with the darker part only. It is noticeable, and expected, that most of the non-zero values fall into the first group. We will extract this group as a vector of values, where each value presents a number of pixels coloured 0-31 in a corresponding bitmap column. Figure 4. shows such vectors of left, right, top and bottom margin, in respective order. For the left margin we see a distinct jump around column 350, where the margin is located. We notice that the right margin follows the same pattern, though we expect more oscillations due to uneven text alignment, as text is not necessarily justified and not all lines end exactly at the margin, but we still notice a jump. Top and bottom margins are hard to distinguish as we cannot be certain if a spike is generated by a line of text or a stain, and instead of a single step, we have multiple small steps as there is empty space between lines. This case is harder to address properly and the method efficiency is within acceptable boundaries with left and right margin only, so we can safely ignore these two cases and focus on left and right margin only. Right margin is symmetrical to the left margin. We can apply the same processing for both margins, as long as we take care of the inverted direction of the right margin.

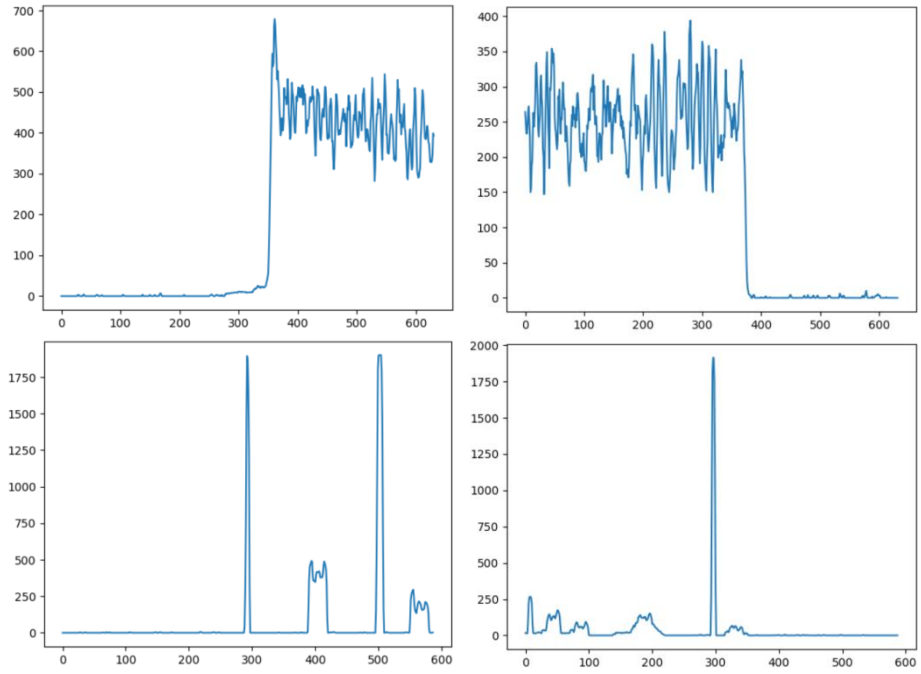

Figure 4. Black color vector line chart of left, right, top and bottom margin

3.1. Step Detection

The problem of finding a margin is now reduced to a problem which is in literature referred to as "step detection", "step finding" and similar. There are plenty of methods which can be applied, like [10]. As we want to find one single, highest, jump it is possible to use the following steps:

1. Smooth the vector using average filter
2. Calculate a vector of neighbour increments using a 16px slide window
3. Smoothen that vector by cutting off small, below average, values
4. Find the leftmost (rightmost) hill

Average filter takes in account five elements, two on the left, two on the right and the current element. Current element is replaced by the calculated average value. Leftmost and rightmost elements use zero to pad the out-of-bounds vector values.

Neighbour increments are calculated differences between two consecutive vector elements. We expect to have the highest value at the position of step. As the page can be tilted up to 16 pixels, it is possible that the step is spread across these 16 bitmap columns/corresponding vector values. For that reason, we use a slide window to calculate the neighbour increment vector. Instead of using only two consecutive vector elements, we will use previous 8 and next 8, and sum their consecutive differences as a value. This has two useful effects: it will cause further flattening of parts with small oscillations; and it will cause accumulation of consecutive jumps and emphasizing the value in margin position in case of a tilted margin.

Figure 5. shows the original vector in blue, smoothened vector in orange and neighbour increments vector in green. We see negative values which imply drops of vector. As we are looking for the highest increment, we can safely ignore these values and treat them as zero.

Margin position is decided by finding the leftmost hill. We expect it to be the highest, but anticipate that it is possible to have two or more such hills, in case of formatted tabular text, hence the leftmost hill is preferred. It is possible that the neighbour increments vector contains lower hils left of margin position. These need to be removed. Simple and efficient method is to find the average hill height and flatten all the hills below the average. Figure 6. shows the smoothened vector in orange and neighbour increments vector with slide window in green, with all the below-average values trimmed to 0.

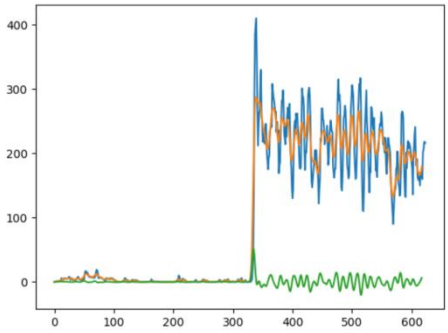

Figure 5. Original vector, smoothened vector and neighbour increments vector

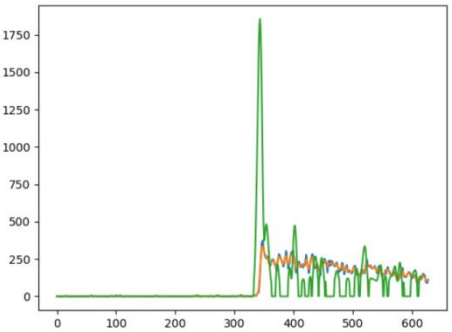

Figure 6. A slide-window neighbour increments vector after trimming

Leftmost hill corresponds to the left margin position. Finding a hill is a simple algorithmic task of iterating through an array and finding the first non-zero value which is greater than its left and right neighbour. Same process follows for right margin. We need to invert the vector and take care to calculate the margin position accordingly, as an offset from right page border.

Accuracy of this method was tested by adapting the code to draw vertical lines on detected margin positions over a set of pages. Manual inspection showed that the method detects margins correctly, with acceptable error rate. In additional calibrating, margins were moved 8 pixels to the outside, to ensure that margin area doesn't contain edges of letters.

4. NOISE RECOGNITION

When present, noise is spread across the entire page with more or less density and granulation. Noise samples vary from a single black pixel to a small group of pixels. Pixel colors vary from black to grayish, but match the previously used first group of quantized colors, with pixel color ranging 0-31. Lighter pixels do not affect OCR efficiency and can be safely ignored. Noise density varies from sample to sample, but on a single sample it is constant.

By applying the previously described method for margin detection we are able to extract two parts of a document which are (mostly) empty. In an empty area noise is easy to detect by simply counting black pixels and setting a threshold. Problem with threshold is that noise density is not the same in all the documents. Setting a high threshold would produce false negatives in documents with low noise density, while setting a low threshold would produce false positives in pages where there is some hanging text or stains. Another problem which we want to address are stains, which should not be counted as noise, as that would distort the process.

By analysing the documents we found the following characteristic of noise: noise is mostly a group of pixels in the form of a dot of varying size, up to 20 pixels. As measuring sizes of each point would be an "expensive" task in terms of execution time, we defined a metric based on contrast and found a threshold value based on statistics of a sample set of documents. A side effect of this method is that it successfully ignores large stains, as these become statistically irrelevant, and thus avoids distortion by this type of noise.

4.1. Measuring Pixel Contrast

Pixel contrast can be expressed and measured in several ways. We have a grayscale image with colors quantized to 8 values which can further be simplified to a monochrome image, as we consider 0-31 as black and the rest as non-black. Contrast of a single pixel is measured by counting how many of 8 surrounding pixels are of the same color. By such measure, a single black pixel on a white background has the highest contrast score of 8. A white pixel on white background has the lowest contrast score of 0, the same as black pixel on a black background – a pixel inside a noisy point or a stain.

Figure 7 shows a zoomed-in part of the margin with noise samples and the same margin after measuring contrast. Contrast values are scaled to match the grayscale spectre, so 0 maps to black and 8 to white. We see that the proposed measure emphasizes edges of noise samples, as this is the area with high contrast.

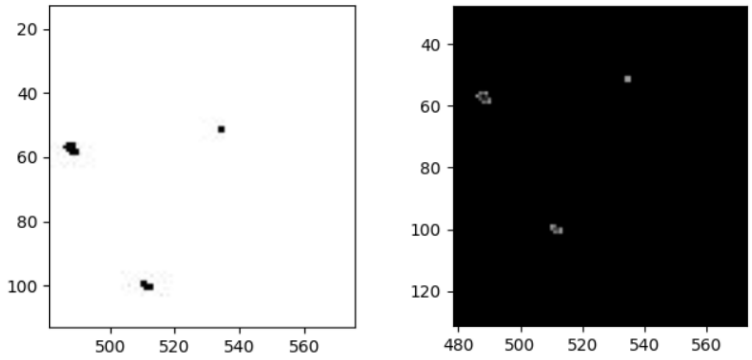

Figure 7. (a) Sample of margin (b) Corresponding pixel contrast values

4.2. Calculating the Noise Granulation Threshold

Margin dimensions vary around 3500x300px, with roughly 1.000.000 pixels in a margin. Noisy documents feature several hundreds of noise samples. Number itself varies and cannot be considered a candidate for a threshold value. Presence of black pixels is an obvious parameter, and we want to determine if black pixels are grouped into small dots, typical to noise, or a large stain which can be ignored as it is not a case of impulse noise.

Size of a stain dictates the ratio of pixels on the edge and pixels inside the stain. Edge pixels have a higher contrast score, while inner pixels have a contrast score of 0. As noise stains are mostly round we expect that the ratio of edge vs. inner pixels be around 0.5. In case of a noise featuring only single black pixels, the ratio is 1, as 100% of pixels are contrasting, edge, pixels. In case of a noise featuring one large stain, the ratio would be low, as the majority of black pixels are inside the stain. We conclude that the smaller the stain the higher the ratio.

To find the ratio threshold we will process a manually separated set of noisy documents. Processing is performed in the following manner: for each document in sample set detect margins; quantize margins to monochrome, by treating 0-31 as black and 32-255 as white; count black pixels; count contrasting pixels; and calculate the ratio as number of contrasting pixels divided by number of black pixels. Results of this process, applied on a sample set, are listed in Table 1. We see that the average ratio is roughly 0.6, which means that the majority of granulation is such that 60% of pixels are on the edge and 40% on the inside of a stain. Ratio of 0.4 is selected as a threshold value of declaring that a margin is filled with the noise in question.

Table 1. Black to contrast pixels ratio in margins of a test set

Margin	Value	Min	Average	Max
Left	Black pixels	1	1328.36	13134
	Contrasting pixels	1	841.52	7862
	Contrasting/black pixels ratio	0.24	0.78	1.00
Right	Black pixels	14	10477.92	66678
	Contrasting pixels	14	6198.08	40126
	Contrasting/black pixels ratio	0.42	0.76	1.00

5. THE NOISE DETECTION METHOD

In 3. we defined the method for detecting left and right margin. In 4. we defined the method to test whether an empty area contains noise. In this specific set of documents margins can be considered as empty, text-free, areas and used to test if document is noisy. Nature of noise is such that the noise, if present, is uniformly spread across the document. Hence, we expect that both margins have similar noise density and granulation. Margins are not the same width, right margin is usually narrower, so margin width ratio needs to be taken in account. We used the ratio of black pixels to all pixels in a sample. Although expected to be approximately the same, experiments showed that this ratio is mostly between 1.1 – 1.7 but varies up to 3. Reason for this could be gradual noise granulation, where granulation decreases from left to right due to the nature of the error source, which is hard to notice and confirm in visual inspection. Experimental results, with testing over the same set where ratio in condition was set to values in range [1,10] showed that the best error-rate is achieved by setting this ratio upper limit to 3, and that for values > 3 it doesn't introduce any changes to method accuracy.

Additionally, we expect a specific type of noise, large stains, to appear in the significant number of documents. This type of noise is to some extent ignored in noise recognition methods, as explained above. To further improve the method we split the margins into quarters and each quarter is processed separately. That way we have four sets of parameters for each margin. Margin sections are sorted and only the middle one in each margin is used for reference. Experiments showed better performance in terms of false positives and false negatives after this step was added, as this step effectively bypasses areas with large stains, if such are present.
The obvious prerequisite for noise presence is that both margins contain black pixels. Further analysis of the dataset showed that samples with less than 12 dark pixels in any of the margins are not to be considered noisy, as the noise is sparse enough that it doesn't interfere with OCR.

Following the noise recognition method previously described, the ratio of contrasting pixels to black pixels has to be at least 0.4.

Considering all these conditions, the noise detection method and the following helper functions pseudocode have the following form:

```
function isBlack(bitmap, x,y): return (bitmap[x,y] < 32)
function contrast(bitmap, x, y):
   contrastedNeighbours = 0
   for i in (-1,0,1):
     for j in (-1,0,1):
       if x+i>=0 and x+i<bitmap.width and y+j>=0 and y+j<bitmap.height
         and isBlack(bitmap,i,j) != isBlack(bitmap, x+i, y+j) :
           contrastedNeighbours++
   return contrastedNeighbours
function findMargins(bitmap):
   marginWidth = round(bitmap.width / 5)
   leftMarginSpectre = [0] * marginWidth
   rightMarginSpectre = [0] * marginWidth
   for j=0 to bitmap.height:
     for i=0 to marginWidth:
       if isBlack( bitmap, i, j): leftMarginSpectre[i]++
     for i= (bitmap.width-marginWidth) to bitmap.width:
       if isBlack( bitmap, i, j): rightMarginSpectre[i]++
   rightMarginSpectre = reversed(rightMarginSpectre)
   leftMargin = findStep( leftMarginSpectre, marginWidth ) - 8
   rightMargin = bitmap.width - findStep( rightMarginSpectre, marginWidth ) + 8
   return (leftMargin, rightMargin)
function findStep( spectre, n ):
   spectre[-2] = 0, spectre[-1] = 0, spectre[n] = 0, spectre[n+1] = 0
   for i=0 to n:
      spectre[i]=(spectre[i-2]+spectre[i-1]+spectre[i]+spectre[i+1]+spectre[i+2]) / 5
   increments = [0] * n
   incrementsSum = 0, prev8sum = 0, next8sum = 0
   for i=8 to n-8:
      prev8sum += spectre[i]
      prev8sum -= spectre[i-8]
      next8sum += spectre[i+8]
      increments[i] = next8sum - prev8sum
      incrementsSum += increments[i]
   incrementAverage = incrementsSum / (n-16)
   for i=8 to n-8:
      if increments[i] < incrementAverage:
         increments[i] = 0
   for i=1 to n-1:
      if increments[i-1]<increments[i] and increments[i]>increments[i+1]: return i
   return n
function isNoisy(pageBitmap):
   (marginLeft, marginRight) = findMargins(pageBitmap)
   left, right= [ (0,0), (0,0), (0,0), (0,0) ]
   for j=0 to pageBitmap.height:
      for i=0 to marginLeft:
```

```
        if isBlack(pageBitmap, i,j) :        left[ floor(j/4) ][0]++
        if contrast(pageBitmap, i, j) > 0 : left[ floor(j/4) ][1]++
    for i=marginRight to pageBitmap.width:
        if isBlack(pageBitmap, i,j) :        right[ floor(j/4) ][0]++
        if contrast(pageBitmap, i, j) > 0 : right[ floor(j/4) ][1]++
sort( left,   lambda a,b: a[0]<b[0] )
sort( right, lambda a,b: a[0]<b[0] )
midBlackPixelsLeft, midBlackPixelsRight = left[1][0], right[1][0]
midContrastingToBlackRatioLeft   = left[1][1] / left[1][0]
midContrastingToBlackRatioRight = right[1][1] / right[1][0]
ratioDifference = midContrastingToBlackRatioLeft / midContrastingToBlackRatioRight
if ratioDifference < 1: ratioDifference = 1/ratioDifference
if  left[1][0] > 12  and  right[1][0] > 12
    and midContrastingToBlackRatioLeft > 0.4
    and midContrastingToBlackRatioRight > 0.4
    and ratioDifference < 3:
      return true
else
    return false
```

5.1. Noise Detection Method Complexity

Pseudocode above initially performs one pass over the marginal area, which is set to be 1/5 of the page on the left and right side. After that pass, a color spectre vector is formed and we have four iterations over that vector and a generated increments vector of the same size. After margins are found, we iterate over both margins to calculate needed pixel counts and generate two vectors with four elements each. Sorting these vectors is considered constant, as four elements can be sorted using a fixed set of nested if conditions.

Let's say we have a bitmap of size nxm. Initial passes take 2*n*m/5 iterations. Then we have 2*4*m/5 iterations over spectre vectors. Finally we have $n*m_l+n*m_r$, where m_l is left margin width and m_r is right margin width. In the worst case both can be m/5, though we expect less in general case. This gives us 2*n*m/5 + 2*4*m/5 + 2*n*m/5 = 4/5*n*m + 8/5*m. Dominating factor here is 4/5*n*m, where we see that order of the method complexity is 80% of page size. In 2. Prerequisites, we calculated the upper boundary as D<0.97*R, where R is the noise removal process. Noise removal process inherently iterates over entire page so order of complexity of R is n*m. As 0.8*n*m < 0.97*n*m we conclude that the proposed method provides the required efficiency.

6. EXPERIMENTAL RESULTS AND DISCUSSION

Accuracy of the proposed method was tested on a manually curated set of actual documents. Test set featured two classes of documents: noisy samples and clean samples. We selected 806 samples, of which 301 were noisy and 505 were clean. Both sets were processed and noise detection results were compared to expected results. This enabled us to count false positives and false negatives. Results are presented in Table 2. We see 7% false negatives and 0.8% false positives. We consider these results acceptable, as we are more concerned to not denoise clean documents, as it introduces errors, while denoising noisy documents is considered an advantage, whatever the denoising rate is. To get the real error rate we need to take in account that noisy documents take 3% of overall documents, which means that the effective error rate is 7% of 3%.

Adjusted error rates are also included in Table 2. and we see that overall error rate is below 1%, which we consider a satisfying result.

Table 2. Test results

Set	Samples	Correct		Incorrect		Adjusted error rate	
		Count	%	Count	%	Occurrence	Adjusted %
Noisy	301	280	93%	21	7%	0.03	0.21%
Clean	505	501	99.2%	4	0.8%	0.97	0.77%
Total	806	781	96.9%	25	5.1%		0.98%

Horizontal table borders and lines in header and footer that are outdented and start inside the margin area proved to be a problematic feature in margin detection, notably on the detection of the left margin. Margin detection relies on color spectre and step detection. As text is, by rule, left-aligned, highest step will most often appear at the place where text starts. Hanging parts of borders and lines will, in that case, fall into margin part of the spectre, and will be treated as noise. Such case could skew the noise detection method and produce a false positive result. Test proved this behaviour and the effect to overal error rate was significant. Line removal is a required step in OCR preprocessing. The order of preprocessing steps doesn't affect the efficiency of implementation, so it was decided to apply line removal before the noise detection step. If used in general case, our noise detection method would have to include line removal filter as a first step and add the filter's complexity into noise detection method complexity.

Noise detection method was adjusted so to use left margin only. Initial tests provided good results in terms of low error rate. As this approach processes 50% less page area than initial, future direction of work will be focused on tweaking the noise detection method so to get satisfying error rate and reduce the processing time per page as this is a critical factor in method's design.

7. CONCLUSIONS

This paper introduced a method of noise detection in a specific surrounding. Project constraints, with large scale document processing being most important, affected the method design in such a way that we focused on achieving the efficiency through adapting to the dataset in question. As a result we got an optimised, efficient solution which satisfies project requirements.

Proposed method works in two stages. First stage detects margins by analysing color spectre of the document. We successfully tweaked the process to reduce the number of iterations over a document and constructed the entire process using elementary operations, which further improves code efficiency. Following the nature of the documents, we were able to reduce the color spectre to one-dimensional vector and perform analyses on that vector in a very efficient manner. Second stage detects presence of noise in margin area. By introducing the contrast metric we managed to perform noise detection in the environment with variable noise granulation and density.

Overall complexity of the method is within acceptable boundaries. Success rate was measured on a manually curated set of documents. Results showed a satisfying accuracy.

ACKNOWLEDGEMENTS
We thank Resolution Technology Ltd. (www.companydatashop.com) for providing curated datasets and supporting the open knowledge.

REFERENCES

[1] Harris, C.R., Millman, K.J., van der Walt, S.J. et al. (2020) "Array programming with NumPy", Nature 585, 357–362. doi:10.1038/s41586-020-2649-2

[2] Verma, Rohit & Ali, J.. (2013). "A comparative study of various types of image noise and efficient noise removal techniques", International Journal of Advanced Research in Computer Science and Software Engineering. 3. 617-622.

[3] Boyat, Ajay Kumar & Brijendra Kumar Joshi (2015) "A review paper: noise models in digital image processing", Signal & Image Processing: An International Journal (SIPIJ) Vol.6, No.2

[4] Saxena, Chandrika, and Deepak Kourav (2014) "Noises and image denoising techniques: a brief survey", International journal of Emerging Technology and advanced Engineering Vol.4, No.3

[5] Sood, Deepika, Anureet Kaur, and Kaushik Adhikary (2013) "A Survey on Despeckling Methods", International Journal of Emerging Technology and Advanced Engineering Vol.3, No.7

[6] Kaur, Jappreet, Jasdeep Kaur, and Manpreet Kaur (2011) "Survey of despeckling techniques for medical ultrasound images", International Journal of Computer Technology and Applications Vol.2, No,4

[7] Karanje, U. B., Dagade, R. (2014) "Survey on text detection, segmentation and recognition from a natural scene images", International Journal of Computer Applications Vol.108, No.13

[8] Agrawal, S.D.,Kullioli, V.C. (2018) "Survey of text detection methods in scene images", EPRA International Journal of Research and Development Vol.3, No.11

[9] "How to remove all the detected lines from the original image using Python?," Stack Overflow, 16-Sep-2019. [Online]. Available: https://stackoverflow.com/questions/57961119/how-to-remove-all-the-detected-lines-from-the-original-image-using-python/57963719#57963719. [Accessed: 28-Jun-2021].

[10] Basseville, M., & Benveniste, A. (1983). "Design and comparative study of some sequential jump detection algorithms for digital signals", IEEE Transactions on Acoustics, Speech, and Signal Processing, 31(3), 521–535. doi:10.1109/tassp.1983.1164131

AUTHORS

Petar Prvulović is a teaching assistant and doctoral student at Union University, School of Computing and a consultant and software developer working mainly in the Justice and Education sector.

Jelena Vasiljević is a professor at School of Computing, Union University, Belgrade, Serbia.