Document Image Applications

Dan S. Bloomberg and Luc Vincent

Google

1 Introduction

The analysis of document images is a difficult and ill-defined task. Unlike the graphics operation of rendering a document into a pixmap, using a structured page-level description such as pdf, the analysis problem starts with the pixmap and attempts to generate a structured description. This description is hierarchical, and typically consists of two interleaved trees, one giving the physical *layout* of the elements and the other affixing *semantic* tags. Tag assignment is ambiguous unless the rules determining structure and rendering are tightly constrained and known in advance.

Although the graphical rendering process invariably loses structural information, much useful information can be extracted from the pixmaps. Some of that information, such as skew, warp and text orientation detection, is related to the digitization process and is useful for improving the rendering on a screen or paper. The layout hierarchy can be used to reflow the text for small displays or magnified printing. Other information is useful for organizing the information in an index, or for compressing the image data. This chapter is concerned with robust and efficient methods for extracting such useful data.

What representation(s) should be used for image analysis? Empirically, *a very large set of document image analysis (DIA) problems can be accurately and efficiently addressed with image morphology and related image processing methods.* When the image is used as the fundamental representation, and analysis (decisions) are based on nonlinear image operations, many benefits accrue: (1) analysis is very fast, especially if carried out at relevant image scales; (2) analysis retains the image geometry, so that processing errors are obvious, the accuracy of results is visually evident, and the operations are easily improved; (3) alignment between different renderings and resolutions is maintained; (4) pixel labelling is made in parallel by neighbors; (5) sequential (e.g., filling) operations are used where pixels can have arbitrarily long-range effects; (6) pixel groupings are easily determined; (7) segmentation output is naturally represented using masks; (8) implementation is simplified because only a relatively small number of imaging operations must be implemented efficiently; (9) applications can use both shape and texture, at multiple resolutions, to label pixels; and (10) the statistical properties of pixels and sets of pixels can be used to make robust estimation.

Table 1 depicts *document image analysis* (DIA) as occupying a high to intermediate position in terms of constraints, which depend on the accuracy of the statistical models representing the collection of images. Bayesian statistical models are the most constrained. Analysis is performed by generation from the models, using *maximum a posteriori* (MAP) inference. These techniques have been used for OCR [7] and for locating textlines [6], and can be implemented efficiently using heuristics despite the fact that they require matching all templates at all possible locations [9].

Examples	Constraint	Approach
letterforms	high	Bayesian MAP
page layout	moderate	morphology with params
natural scenes	low	ad hoc

Table 1: *Effect of constraints on the approach to image analysis*

Many DIA problems are not framed in a strict Bayesian format. Although the models are not well-specified, there exist regularities that allow identification of layout parameters (such as average spacing between words and text lines) and, eventually, the layout hierarchy itself. This involves use of both shape and texture, for which morphological operations are ideally suited. At the other extreme, arbitrary natural scenes have very few constraints and continue to defy general attempts at analysis.

The most important low-level operations for DIA fall into five classes:

- *Morphological.* Operations on binary images are by far the most common.
- *Rasterop.* Ubiquitous bit-level operations, these are used for implementing binary morphology, binary logic (e.g., painting and masking) over arbitrary rectangles.
- *Rank reduction.* Nonlinear operations where the subsampled *destination* (dest) pixels are determined using a rank threshold on a tile of *source* (src) pixels, both for binary and grayscale images.
- *Binary reconstruction.* Operations that fill into a mask image from a seed image. These are crucial for accurate segmentation.
- *Connected components.* This differs from the first four operations in that it reads and writes single pixels rather than full words, and can generate non-image data, such as bounding boxes.

These operations can all be implemented efficiently. The first three are parallel: each dest pixel depends only on src pixels. The last two should be done sequentially: the order of operations matters because each dest pixel can depend on previously computed dest pixels. Sequential operations allow a src pixel to affect a dest pixel an arbitrary distance away, whereas parallel operations have a limited extension of influence.

Programs that generate the output shown in the following applications are indicated in the captions **(!!!! actually, not yet: it would be nice to give the links here in the book !!!!)**. Source code for many of the algorithms described here, including all the examples, can be obtained at **(!!!!! Fix this !!!!!!)** *livre-hermes-web-page/some-tar.gz*.

2 Applications

We have space to demonstrate a small number of document image applications that benefit by using a morphological approach.

2.1 Word extraction from a music score

The extraction of words from a music score is very simple. From a 1 *bit/pixel* (bpp) image, a large horizontal morphological erosion generates "seeds" in the staff lines. Then a binary reconstruction (seed fill), using the original image as a mask, recovers the lines and everything touching them. Lyrics and other musical notations are then extracted by XORing with the original.

2.2 Page segmentation

Segmentation is the fundamental operation in DIA. There are many variations and approaches, depending on the goals of the analysis. The goals can be partially specified by the pixel accuracy desired and the cost of various errors. Examples of such goals are:

- Is there an image (or textblock) on the page?
- If there are images (or textblocks), where are they?
- Are there other graphics elements on the page?
- Locate the hierarchical (tree) structure of the text: blocks, paragraphs, sentences, words, characters.
- Assign logical labels to page elements

For a real application, the cost of errors must be considered. For example, if the primary goal is good visual appearance, and the non-image part is quantized into a small number of levels, the cost of identifying image pixels as non-image can be much higher than making the opposite mistake. By contrast, if the goal were to identify all the text as a preprocessing step for OCR, it is much worse to lose text regions than to label some image pixels as non-image.

Page elements can be labeled with binary masks. Each pixel in a binary mask represents a yes/no decision about whether that pixel has a particular label. Pixels can be represented as fg in multiple masks, such as a pixel that is labeled as fg in both a textline mask and a textblock mask. For example, a halftone mask, with fg pixels over pixels in halftone regions, can be used to remove those image pixels before doing text analysis, or to direct an operation to render the image and non-image pixels differently. The latter is often desirable because text is best rendered with high contrast, whereas images are usually rendered with dithering on printers or with many levels on displays to avoid posterization. In the following, we show how to start with an image and progressively filter different regions, using the implicit shape and texture properties.

Let us first show the use of rank reduction to answer the question: Is there an image on the page? Figure 1 shows the sequence of images. Although a sequence of reductions is taking place, the results are all displayed at the same resolution. Starting with a 300 *pixel/inch* (ppi) image containing $8x10^6$ pixels (a), do a cascade of four 2x rank reductions. Parts (b) and (c) show the results at 4x and 16x reduction, using levels 1 and 4 followed by 4 and 3, rsp. A final $5x5$ erosion yields the result (d), and a test for fg pixels gives the answer. This is a computationally inexpensive procedure, taking only 1 msec on a standard 3 GHz processor! This result can be used as a seed in a binary reconstruction to generate the halftone mask, as we now show.

There are several different morphological ways to identify text and halftones. Some involve binary reconstruction to form the masks at some point in the calculation. The images are assumed to be reasonably

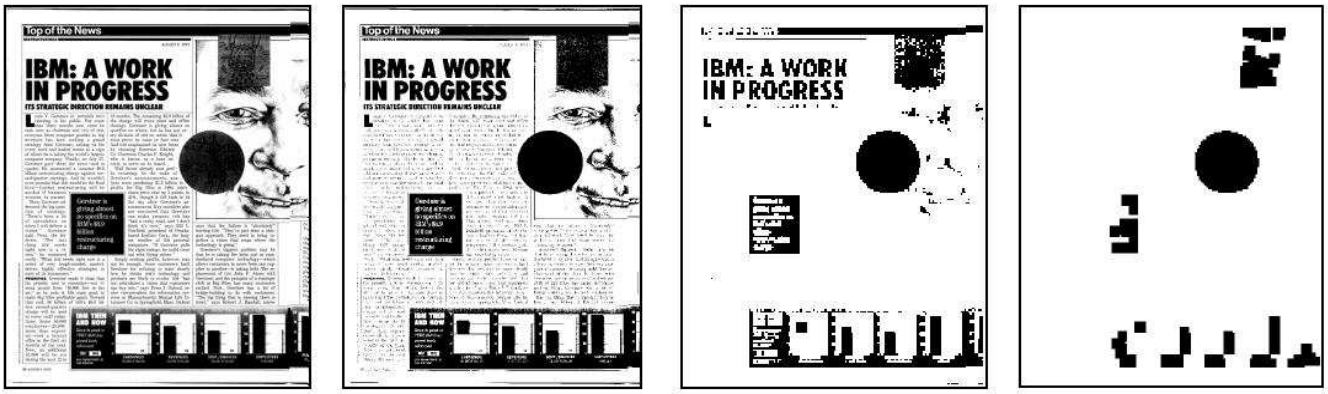

Figure 1: *Generation of halftone seed to identify the existence of images.*

well deskewed. Here is an almost trivial approach: do a horizontal closing followed by a smaller horizontal opening. This can leave pixels within text lines as solid fg rectangles, separated vertically by bg pixels, and pixels within halftone regions as solid fg. This is the essence of an early morphological approach called RLSA [4]. A vertical opening can then remove the text lines, leaving the halftone mask.

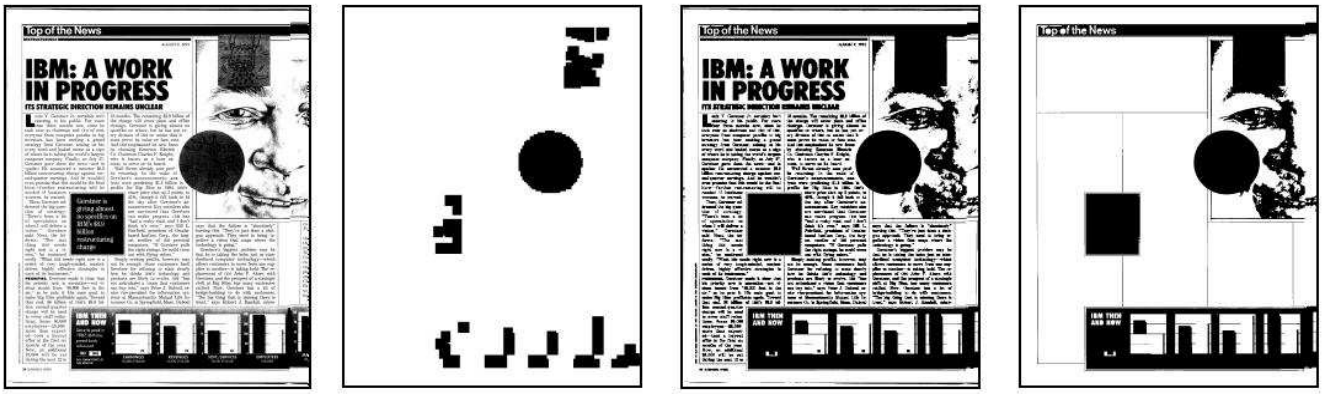

Figure 2: *Generation of halftone mask.*

We now show a somewhat more accurate method for page segmentation. All operations except the halftone seed construction are performed at a resolution of 150 ppi. Start by finding the binary masks that label image pixels. In the following, we show the operations on two different images that have text, image and rules in nontrivial layouts. Figure 2 shows steps in projecting out the halftone parts of the page (a). The seed (b), composed of pixels that are nearly certain to be within the halftone region(s), is generated by a sequence of 2x rank reductions (levels 4, 4 and 3), followed by a $5x5$ opening and 8x replicated expansion back to 150 ppi. This was shown in Figure 1. The clipping mask (c) is designed to connect pixels in each halftone region (so that even a single seed pixel will fill it entirely), but not to form a bridge to any pixels in non-halftone regions. It is generated from (a) using a 2x reduction (level 1) followed by a $4x4$ closing. The halftone mask (d) is then generated by binary reconstruction from the seed into the mask.

The next step is to find the text lines. These can be consolidated through a horizontal closing, but such an operation will join lines in different columns, so a vertical whitespace mask must be generated that can

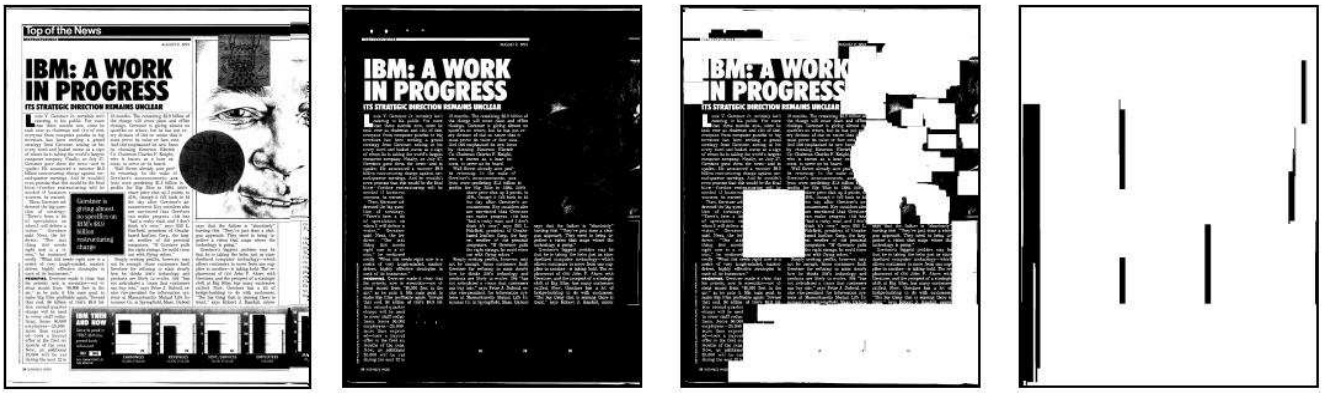

Figure 3: *Generation of whitespace mask.*

later restore the white gutters. This is shown in Figure 3. Starting from the input image (a), subtract the halftone mask and invert the result (b). Opening with a large vertical Sel can leave components that will break text lines with a large amount of white space above or below, but this can be prevented by opening first with a Sel that is wider than the column separations and higher than the maximum distance between text lines (c). After these pixels are removed, open with a $5x1$ horizontal Sel to remove thin vertical lines, followed by opening with a $1x200$ vertical Sel to extract long vertical lines (d).

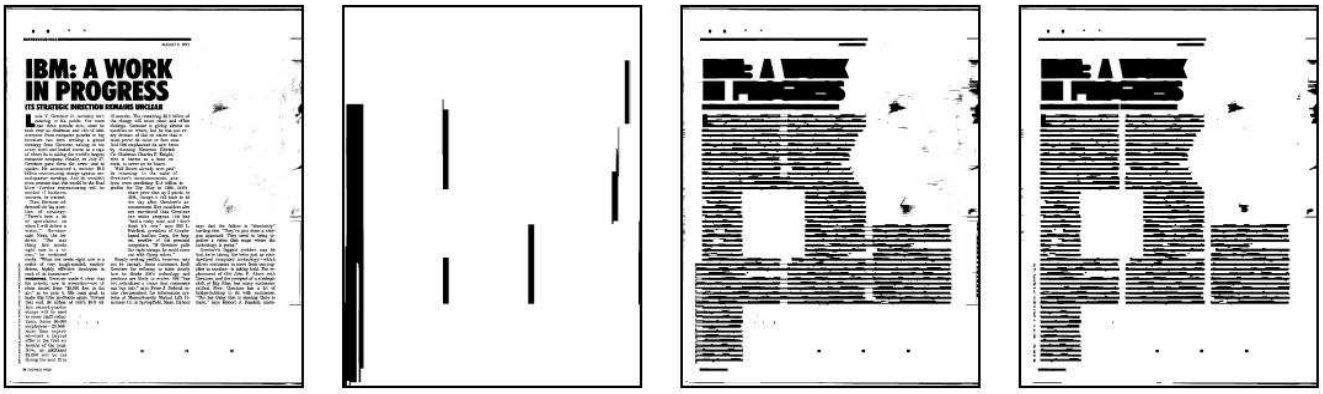

Figure 4: *Generation of textline mask.*

Figure 4 shows the text line extraction process, with the whitespace mask computed in (b). Starting again with the image (a), solidify the text lines using a $30x1$ closing (c). Text in adjacent columns that has been joined is then split by subtracting the vertical whitespace mask, and a small $3x3$ noise-removal opening yields the textline mask (d).

Figure 5 shows the steps taken to consolidate the text blocks. The original page is shown in (a). Begin with the textline mask, and join pixels vertically using a $1x8$ closing (b). Then, for each cc separately, do a $30x30$ closing to form a solid mask. By closing separately, we can use a large Sel without danger of joining separate regions. Follow this with a small $3x3$ dilation, to insure coverage of the mask components. At this stage, some textblock components need to be joined horizontally, and this is done with small horizontal closing (c). Because this closing can join textblocks separated by very narrow gutters (which did not happen in the two examples shown), the vertical gutter mask is again applied to split blocks that may have

Figure 5: *Generation of textblock mask.*

been joined, and small components are removed to obtain the textblock mask (d). This can be further filtered for size and shape.

In these examples of page segmentation, a number of parameters were specified *a priori* for the filter sizes, rather than being computed using measurements on each page. The question naturally arises whether such an open-loop approach is robust. Perhaps surprisingly, the answer is in the affirmative, if by robust we mean that errors where large numbers of pixels are misclassified occur very rarely. The robustness is tested in two ways: (1) by using the algorithm on a large number of pages and (2) by demonstrating the the results are relatively invariant when the parameters are changed by about 30 percent in each direction. The latter is easily measured by scaling the image up and down by this fraction. In this way, it is seen that when computing textblocks on a scaled up image, some of the textlines are not joined, so the vertical closing parameter should be larger. The advantage of this highly-empirical approach is that failures are easy to find and to analyze, and proposed improvements are quickly tested.

2.3 Skew detection

Image deskew greatly simplifies page analysis and improves both the performance of symbol-based compression (*jbig2*) and the displayed appearance of the page. Most approaches to the computation of a global skew for 1 bpp images are based on Hough transforms or pixel projection profiles. Other attempts have used fourier transforms, connected components, and special pre-filterings such as a rosette of morphological pixel correlation filters [10]. For a short description of some of these methods, see [2].

Most of these approaches have difficulty generating an accurate signal from the lines of text in situations where there are multiple, unaligned columns, or the scan includes part of a second page. The simplest and arguably the most effective way to avoid these problems was described by Postl[11] in 1988. Postl maximized the variance of the *difference of pixels on adjacent scanlines*. Let the sum of pixels in the i^{th} scanline be $p_i(\theta)$, where θ is the angle through which the image is rotated (or vertically sheared). Then Postl's signal is

$$S(\theta) = \sum_i (p_i(\theta) - p_{i-1}(\theta))^2 \tag{1}$$

where the sum extends over all scanlines in the image. The image is then deskewed by rotating through the angle θ for which $S(\theta)$ is maximized. This is effective because, when the page is aligned, most of

the signal comes from a relatively small fraction of scanlines; namely, those at the base and x-height of the text lines. Halftone pixels contribute little to such a differential signal. Text lines in each of multiple columns will contribute relatively independently to the signal if they are not aligned. And the peak will be very sharp, corresponding to an angular half-width in radians of approximately 1/(textline width in pixels). At 300 ppi, with a textline width of 1500 pixels, the half-width of the peak in $S(\theta)$ is about 0.04 degrees. This is more than sufficient for visual appearance, because it is unusual to notice image skew that is less than 0.2 degrees. Results have been given on a data set of about 1000 images [1], and these have been compared with a morphologically-based filtering approach [10].

2.4 Text orientation detection

The *hit-miss transform* (HMT) can be used to determine the orientation of Roman text, because there is a preponderence of ascenders over descenders (approximately 3:1 for English). Consider the four hit-miss Sels:

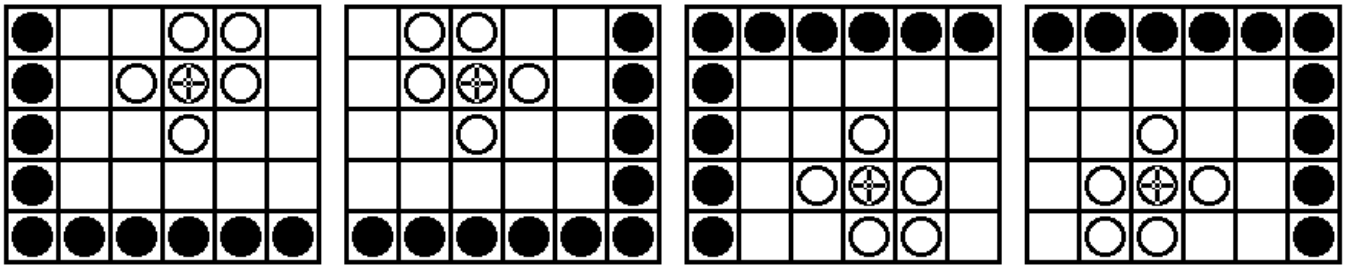

Figure 6: *hit-miss Sels for extracting character ascenders and descenders.*

where the Sel origin is indicated by a small black circle. The signal in this case is the difference between the number of ascenders, identified from the HMT using the first two Sels, and the number of descenders, using the last two Sels. The statistical significance of this difference is determined as follows. The expected variance in each of these numbers is proportional to their square root. The probability that the two populations can be distinguished (i.e., that the distributions do not overlap) is estimated from the square root of the sum of the individual variances:

$$\sigma_o = \sqrt{N_{up} + N_{down}}/2 \tag{2}$$

Then the *normalized orientation signal* is defined as the difference between the number of ascender and descenders, expressed as a multiple of σ_o:

$$\bar{S}_{orient} \quad \equiv \quad | \, N_{up} - N_{down} \, | \, / \sigma_o \quad = \quad 2 \, | \, N_{up} - N_{down} \, | \, / \sqrt{N_{up} + N_{down}} \tag{3}$$

Usually there will be different prior probabilities for the text orientation, so different thresholds are in general set on the normalized signal for a decision to be made. The signal can also be measured in landscape orientation, and the two signals compared, using appropriate priors, to determine the orientation as one of a set of four directions.

Before doing the HMT, the textline structure should be simplified to fill the holes within the x-height region, leaving only the ascenders and descenders. This can be done with a horizontal closing to solidify the text line, followed by a larger opening to remove all ascenders and descenders that have possibly been

joined by the closing. The ascenders and descenders can then be simply reconstructed by ORing with the original image. These pre-HMT operations can usually be done at a lower resolution of between 100 and 150 ppi, using a dilating rank reduction to preserve pixels.

After the HMT we have pixels in small clumps associated with each ascender and descender. To get the ascender and descender count, we can find the number of 8-cc, but an even more efficient and robust way is to do a cascade of 2x rank (level = 1) reductions that consolidates each small cluster into a tiny cc, followed by counting the number of components at this reduced resolution.

2.5 Pattern matching

The ability to do fast pattern matching between elements of document images, such as cc or character or word images, is an important underpinning of many important applications. Some examples are:

- Most OCR systems use image matching with a large library of templates.
- Lossy jbig2 compression of binary images requires unsupervised classification of components into a relatively small number of similarity classes, the templates of which are used to represent each instance of its class when rendering the page.
- The generation of similarity classes can be used to improve the quality of a rendered image, by generating grayscale templates from a set of binary instances. These grayscale templates can be used directly to substitute for the binary instances, or they can be converted to higher resolution binary templates, a process called "super-resolution."
- Hit-miss Sels can be generated automatically from a pattern on an image, and then used to find all other occurences of this pattern.
- Applications such as document image summarization [3] estimate important words, phrases and sentences by the occurrence of repeated word shapes.

Pattern matching requires some way to measure similarity between elements. Two popular similarity measures for binary images are the Hausdorff distance and correlation. Once a measure is chosen, along with a threshold for declaring two patterns sufficiently similar to belong to the same class and a policy (typically "greedy" or "best match") for terminating the search for a matching template, unsupervised matching can proceed [8],[13]. In practice, for small text that is scanned at 300 ppi, character confusion can occur with a Hausdorff distance threshold as small as 1. Consequently, correlation is preferred, as described below.

For the classifier application, we have a set of templates for existing classes and a set of instances yet to be assigned to a class (or, if not assigned, to become the template for a new class). Greedy matching works well: each instance must be matched against the templates until a sufficiently close match is found.

2.5.1 Correlation image comparator

Correlation is computed from the fg pixels, with centroids aligned. Let A and B be the binary images to be compared, and denote the number of fg pixels in an image X by $|X|$ and the number in the intersection of the two images by $|A \cap B|$. A is one of the templates and B is an instance to be classified. Then the correlation is defined to be the ratio

$$C(A, B) = (|A \cap B|)^2/(|A| \times |B|) \tag{4}$$

The correlation is compared with an input threshold. However, because two different thick characters can differ in a relatively small number of pixels, the threshold itself must depend on the fractional fg occupancy of image B. Let the bounding box of B be $w_B \times h_B$. Then the fg occupancy of B is $R = |B|/(w_B \times h_B)$. The modified threshold T' then depends on two input parameters, an input threshold T and a weighting parameter F ($0.0 \leq F < 1.0$):

$$T' = T + (1.0 - T) \times R \times F \tag{5}$$

For 300 ppi images, it is found experimentally that values of T = 0.8 and F = 0.6 form a reasonable compromise between classification accuracy and number of classes.

2.5.2 Component alignment for substitution

A jbig2 encoder must specify, for each instance in the image, the class membership (an index) and the precise location that the template for that class is to be placed by the decoder. Although the correlation matching score is found with centroids aligned, in a significant fraction of instances, the best alignment (correlation-wise) differs by one pixel from centroid alignment. This correction is important for appearance of text, because the eye is sensitive to baseline wobble due to a one-pixel vertical error. It is thus necessary to measure the XOR of the two images at the location where the centroids line up, and at the eight adjacent locations. The best location has the minimum number of pixels in the XOR.

2.5.3 Hit-miss comparator

The HMT is a general filter for matching an arbitrary binary pattern to a binary image. There are no constraints on the content of the pattern fg. However, the characteristics of the hit-miss filter must match the expected variation in the pattern, because the HMT doesn't have a rank parameter: every hit and miss must match. For document images, variation can take the form of boundary noise, salt and pepper noise, rotation, scaling, and other image distortions. As a general rule, it is best to put hits and misses far enough from the boundaries to completely avoid boundary noise. One should avoid using more hits or misses than necessary, because it increases both computation time and the likelihood that an instance is missed. If too few hits or misses are used, false matches will be hallucinated. To reduce sensitivity to small skew and scale changes, the aspect ratio of the pattern should ideally be close to 1.

Of several methods for automatically generating a hit-miss Sel from a pattern, we describe and illustrate the "Boundary" method. Select a fraction of fg and bg pixels that are at specified distances from the boundary. First the fg and bg contours at the specified distances are generated. Then the hits are chosen by subsampling along a traversal of the fg contour, and likewise for the misses. These four parameters allow flexible specification of the hit-miss Sel (`pixGenerateSelBoundary()`).

Figure 7 illustrates a hit-miss Sel generated by the boundary method. The pattern (on top) is reduced 8x and the hits and misses are placed at a distance of 1 from the boundary, with hits subsampled every 6th pixel in the fg and misses every 12th in the bg. The HMT is very fast; on a 25M pixel image, reduced 8x to 400K pixels, it takes about 12 msec.

Using just the "T" in the pattern makes the HMT more robust to skew and to variations in scale. Figure 8 shows the pattern and the Sel generated at 4x reduction. The HMT on the 4x reduced image (1.6M pixels) takes 0.2 sec.

Figure 7: *Pattern and hit-miss Sel generated from it at 8x reduction.*

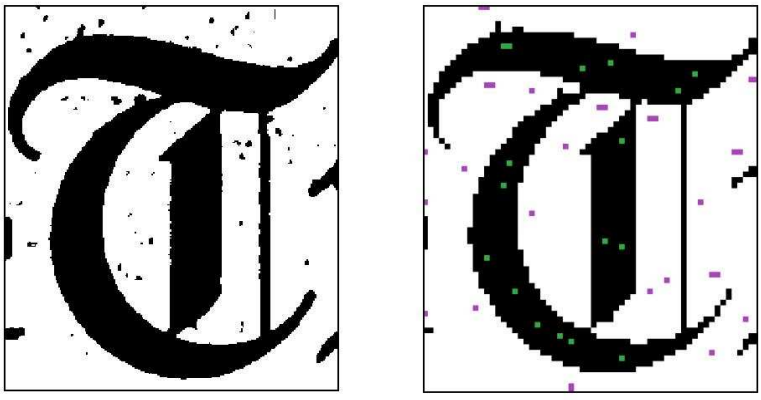

Figure 8: *Pattern and hit-miss Sel generated from it at 4x reduction.*

2.6 Background estimation for grayscale images

We finish with an application showing the use of grayscale morphology. Suppose a document image is captured in grayscale, but with a significant variation in the background illumination across the page. Suppose you wish to render the image in grayscale, but reconstructed as it would appear if the illumination were uniform.

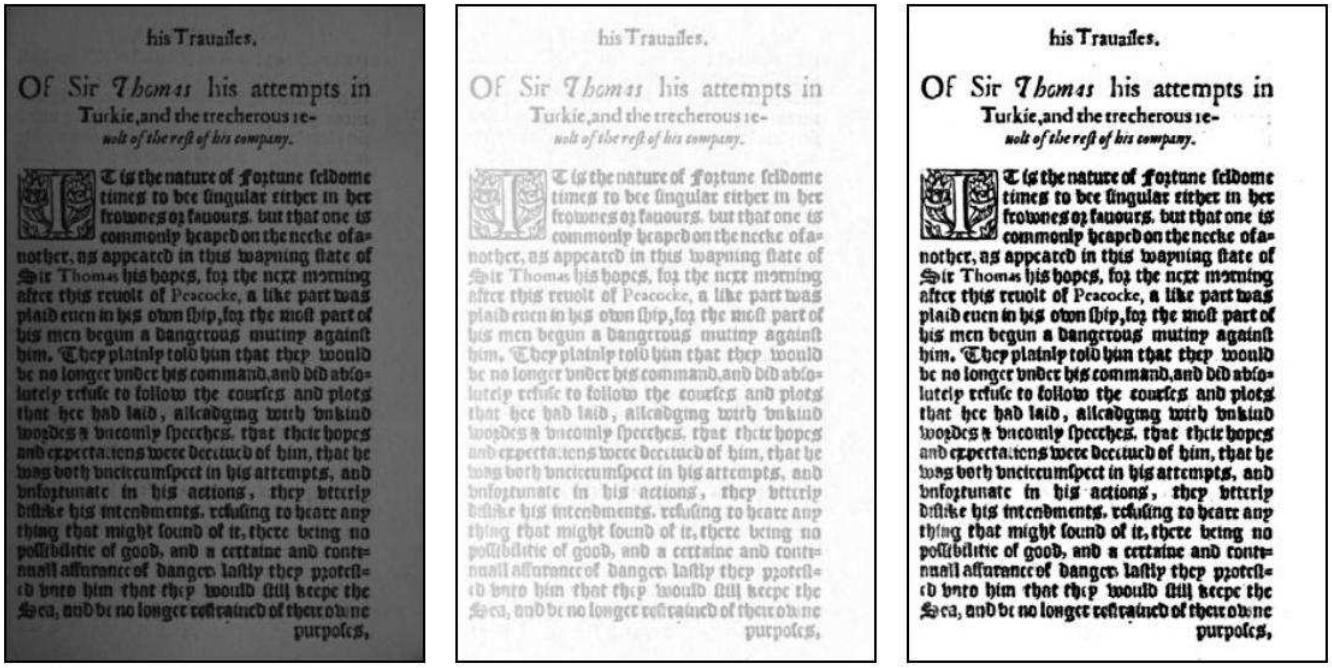

Figure 9: *Use of grayscale tophat to compensate for uneven illumination.*

The morphological tophat is a robust tool. The bg variations can be largely removed by first closing the input image (to remove the fg) and then subtracting the input image from the result. Figure 9 shows the processing sequence, starting with an 8 bpp grayscale page image at a resolution of 150 ppi, in (a), and performing a tophat with a $15x15$ Sel, which is photometrically inverted (b). The closing in the tophat is performed relatively efficiently using the van-Herk/Gil-Werman (vHGW) algorithm[5, 12], separably, which does the closing in a time independent of the size of the Sel.

The result (b) has a washed-out appearance because the input image (a) has a very dark bg. The appearance can be improved by using a linear *tone reproduction curve* (TRC) to increase the dynamic range, giving (c). In this case, we mapped pixels in (b) with values below 200 to 0 and pixels with values above 245 to 255. The value 245 is chosen for the white point to eliminate most of the bleedthrough from the other side of the page. Nevertheless, some deficiencies remain, as the background is not entirely cleaned and the text on the left side of the page is somewhite lighter than the rest.

Why not simply binarize with an adaptive threshold on (a)? There are two reasons. First, by mapping to a grayscale image, we give ourselves the option to change the gamma and the dynamic range of the image before thresholding. Second, we preserve the option of retaining the mapped grayscale image, which displays better on a screen that supports anti-aliasing.

References

[1] D. S. Bloomberg and G. E. Kopec and L. Dasari, "Measuring document image skew and orientation," *SPIE Conf. 2422, Doc. Rec. II*, pp. 302–316, 1995.

[2] D. S. Bloomberg, "Analysis of document skew," *http://www.leptonica.org/papers/docskew.pdf*.

[3] F. R. Chen and D. S. Bloomberg, "Summarization of imaged documents without OCR," *CVIU*, Vol 70, No 3, pp. 307-320, 1998.

[4] K. Wong, R. Casey and F. Wahl, "Document analysis system," *IBM J. Res. Develop*, **26**(2), pp. 647–656, 1982.

[5] J. Gil and M. Werman, "Computing 2-D min, median and max filters," *IEEE Trans PAMI* **15(5)**, pp. 504-507, May 1993.

[6] A. Kam and G. Kopec, "Document image decoding by heuristic search," *IEEE Trans. PAMI* **18**, pp. 945–950, Sept. 1996.

[7] G. Kopec and P. Chou, "Document image decoding using Markov source models," *IEEE Trans. PAMI* **16**, pp. 602–617, June 1994.

[8] A. G. Langley and D. S. Bloomberg, "Google Books: Making the public domain universally accessible," *SPIE Conf. 6500, Document Recognition and Retrieval XIV*, paper 6500-16, 2007.

[9] T. P. Minka, D. S. Bloomberg and A. Popat, "Document image decoding using iterated complete path search," *SPIE Conf. 4307, Document Recognition and Retrieval VIII*, pp. 250–258, 2001.

[10] L. Najman, "Using mathematical morphology for document skew estimation," *SPIE Conf. 5296, Document Recognition and Retrieval XI*, pp. 182–191, 2004.

[11] W. Postl, "Method for automatic correction of character skew in the acquisition of a text original in the form of digital scan results," *U.S. Pat.* 4,723,297, Feb. 2, 1988.

[12] M. van Herk, "A fast algorithm for local minimum and maximum filters on rectangular and octagonal kernels," *Patt. Recog. Letters*, **13**, pp. 517-521, 1992.

[13] `www.leptonica.org`