Morphological Operations for Digital Image Text Classification

Thalles Santos Silva*
Hélio Pedrini†

Abstract

Given a binary image from a digitalized text document, how can we localize and count the number of lines and words in the document? This piece proposes a method to localize lines and their respective words in digitilized text documents. The method is based on digital imagery morphological operations. We describe and show the method's result along with an explanation of the solution.

1. Configuração

This project was developed using Python version 3 and Jupyter Notebook. The libraries used across the project are NumPy [1] and OpenCV [2] to do image-based computation and matplotlib [3] for visualization. The Notebook is divided into subsections analogous to the sessions contained in this document.

2. Intruduction

2.1. Morphology

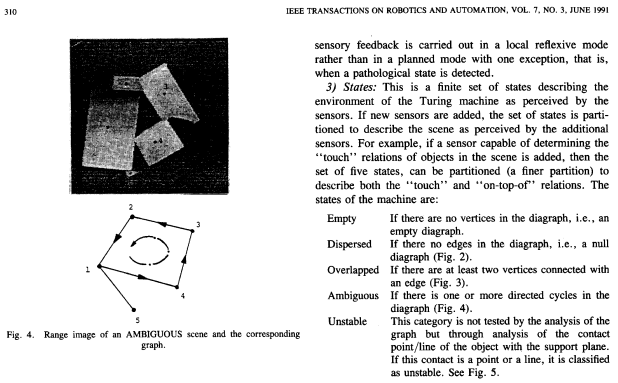
A morphological operation is a set of image processing algorithms to process image pixels using a pre-defined kernel. This kernel, known as a structuring element, defines a pattern that will be used to process images based on its shape.

The most common morphological operations are dilation and erosion. In both methods, a pre-defined kernel convolves the image and substitute the target (center) pixel following a criterion.

Intuitively, erosion has the effect of deteriorating the boundary pixels of foreground elements. We can think of it as expanding the background pixels and reducing the shape of foreground objects. In short, a pre-defined kernel is passed over the image as a 2D convolution. Then, assuming a binary image, an input pixel will be 1 if all the pixels under the kernel is 1. Otherwise, it gets a 0.

*Is with the Institute of Computing, University of Campinas (Unicamp). **Contact**: thalles753@gmail.com

†Is with the Institute of Computing, University of Campinas (Unicamp). **Contact**: helio@ic.unicamp.br



310 IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION, VOL. 7, NO. 3, JUNE 1991

Fig. 4. Range image of an AMBIGUOUS scene and the corresponding graph.

sensory feedback is carried out in a local reflexive mode rather than in a planned mode with one exception, that is, when a pathological state is detected.

3) States: This is a finite set of states describing the environment of the Turing machine as perceived by the sensors. If new sensors are added, the set of states is partitioned to describe the scene as perceived by the additional sensors. For example, if a sensor capable of determining the ''touch'' relations of objects in the scene is added, then the set of five states, can be partitioned (a finer partition) to describe both the ''touch'' and ''on-top-of'' relations. The states of the machine are:

Empty	If there are no vertices in the diagraph, i.e., an empty diagraph.
Dispersed	If there no edges in the diagraph, i.e., a null diagraph (Fig. 2).
Overlapped	If there are at least two vertices connected with an edge (Fig. 3).
Ambiguous	If there is one or more directed cycles in the diagraph (Fig. 4).
Unstable	This category is not tested by the analysis of the graph but through analysis of the contact point/line of the object with the support plane. If this contact is a point or a line, it is classified as unstable. See Fig. 5.

Figure 1. Original binary input image of a digitilized text document.

We can also think of erosion as applying a minimum function over the pixels under the kernel. In this way, the output pixel will be the smallest value among all the pixels under the kernel. As we will see shortly, erosion has also the effect of removing white noise.

Dilation acts in the opposite way. It expands the boundaries of foreground elements. In this situation, we can think of it as a transformation that expands the foreground elements. As a consequence, it also reduces the number of background pixels. In terms of structuring elements, as oppose to erosion, given an input pixel, the output is 1 if at least one of the input pixels (under the kernel) is also 1. In the same way, the output pixel will be the maximum value among all pixels that fall within the kernel.

We can also combine these 2 operations. For instance, Opening is the application of erosion followed by dilation. Similarly, dilation followed by erosion is called Closing. Intuitively, Opening is widely used as a noise removal transformation. Closing, on the other hand, is used to get rid of small gaps within the foreground objects.

2.2. Finding and Counting Lines

To find document lines, we followed a series of morphological operations. In order to allow the morphological operators to act as expected, we transform the input image pixels to 0s or 1s only. Then, the pixel values are inverted to

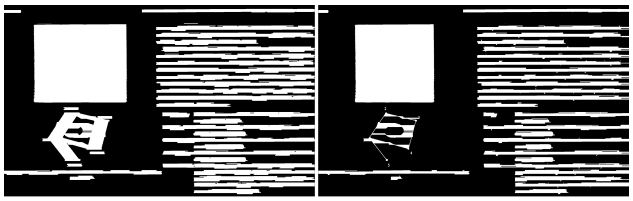


Figure 2. Horizontal dilation (a) followed by erosion (b) on the input image 1

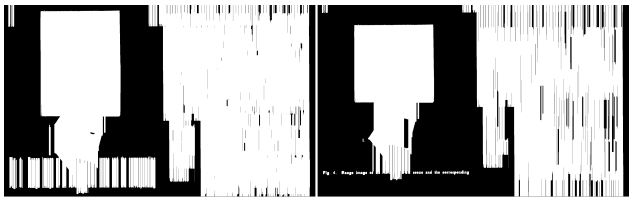


Figure 3. Vertical dilation (a) followed by erosion (b) on the input image 1

guarantee 2 properties:

- Black Pixels (0s) represent the background.
- White Pixels (1s) represent the foreground.

The first step is to create a structuring element with 100 pixels wide and 1 pixel tall. Using this element, we apply a dilation operation to the input image. We expect the dilation to increase the boundary pixels of the foreground elements (the words in the image). Plus, the elongated shape of the kernel (100 pixels wide) has the effect of stretching the foreground pixels in the horizontal direction. As a result, the pixels representing the words in the text gets merged as one wide block of foreground pixels. Figure 2 (a) shows the result.

Next, we erode the result from dilation using the same structuring element. The goal is to refine the boundaries of the blocks that represent the foreground. Here, erosion removes a few pixels from the boundaries of each block. As a result, we get smoother blocks as shown in Figure 2 (b).

This first series of operations allowed us to find blocks of foreground pixels in the horizontal direction. Now, we are going to do the same operations but in the opposite direction.

We start off by creating a different structuring element. **This time, it has 200 pixels tall and only 1 pixel wide**. When we dilate the original image using this kernel, we get a similar effect from the first dilation. However, the blocks of foreground elements that get merged are in the vertical direction. Again, after dilation, we apply an erosion operation so that we can get finer results. Figure 3 displays the results of vertical dilation and erosion.

The intermediate results representing horizontal and vertical blocks can now be merged using an AND pixel-wise operation. The idea is to consolidate the findings in both directions. Intuitively a valid line of text has to appear in both representations to be considered a valid line. As we might expect, this step removes many candidates that were only detected horizontally or vertically in the input image. Since we represent foreground pixels as 1 (white) and background pixels as 0 (black), a pixel-wise AND operation can be expressed as simple multiplication.

Lastly, merging the vertical and horizontal representations might add some holes inside the foreground objects. To make sure we minimize this problem, we apply a Closing operation with a **kernel shape of 30 pixels wide and 1 pixel tall**. Figure 4 shows the final result after Closing.

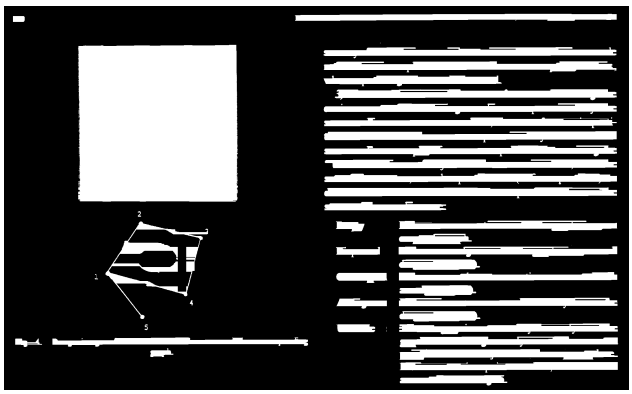


Figure 4. Result of the morphological operations. Each blob (connected component) represents a line of text.

Next, we want to find the bounding boxes surrounding the connected components in the resulting representation. A connected component is a set of pixels that share a common pre-defined rule. To find such regions we used the OpenCV *connectedComponentsWithStats()* function.

Given a binary image, this function returns a segmented image where the pixels of each component are labeled with a unique value. In other words, the pixels of the first component may have values 1, pixels of the second component may be 2, and so on. Also, for each component, it returns the smallest rectangle that fits that component, also known as the bounding boxes.

Once we have the bounding boxes of each component, we are going to use the pixel information (within the boxes) to calculate features. In turn, we will use these features to design a set of rules to classify a given component as text or non-text. For each bounding box, we are going to calculate 2 features.

- The ratio of black pixels and the total number of pixels in the box (pixel ratio).

- The ratio of white to black transitions and the total number of pixels in the box.

The idea is that these features will be very similar for components that represent lines of text. On the other hand, other components will express a different distribution of these features. Thus, we can design heuristics that classify a component as text or non-text by comparing feature values. The final result is shown in Figure 5. Also, Figure 6 shows some of the connected componets classified as text by the algorithm.

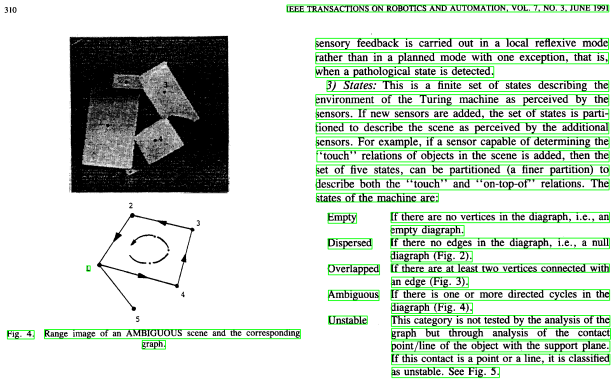


Figure 5. The algorithm was able to count 35 lines. Each line is surrounded by a bounding box.

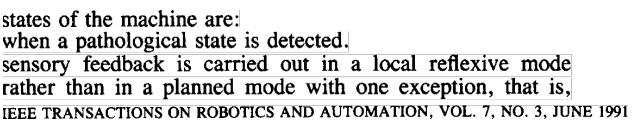


Figure 6. Examples of lines recognized and classified as text.

As we can see, the results in Figure 5 are very robust. Note that some lines in the text got spit in 2. That is because, in certain points, some words are very far apart from each other. As a result, the horizontal dilation wasn't able to merge them into a unique block. Also, because of this extra gap, the vertical operations weren't able to detect them as a single entity.

Considering lines of text as document lines (regardless of the spacing among words), the document has 28 lines. Nevertheless, the algorithm was able to count a total of 35 lines.

2.3. Finding and Counting Words

To localize individual words in the text, the process is very similar. The major difference lies in the shape of the structuring elements.

For the first set of operations, **we choose a 12x1 (12 pixels wide 1 pixel tall) structuring element**. Note that we reduced the horizontal reach of this kernel (compared to the one used to identify lines). The idea is that, while most words have an elongated (rectangular) shape, it is not as wide as lines are.

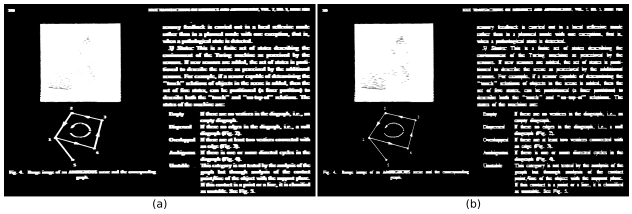


Figure 7. (a) Dilation and (b) Erosion using a narrower structuring element in the input image 1. Note how characters (of the same words) get merged in a common block.

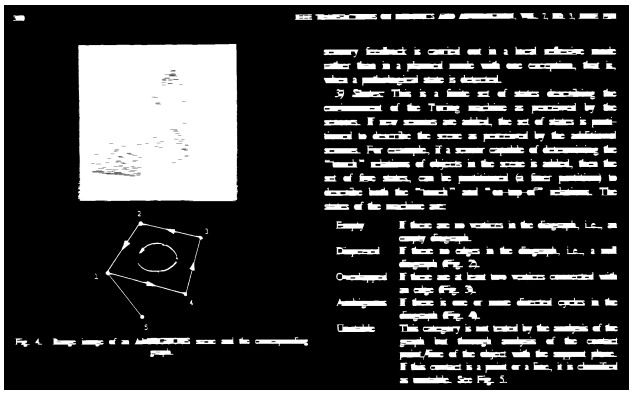


Figure 8. Final word morphological processing representation. Each connected component represents a word in the text.

Here, dilation will expand the foreground pixels in the horizontal direction and erosion will refine the new blocks of text. Because we chose a narrower structuring element, dilation has the effect of merging letters of the same word in a unique block. Figure 7 shows the resulting image.

To find vertical patterns, we define a **structuring element with a 1-pixel width and 4 pixels height**. Again, we take the original input image and apply dilation and erosion using this kernel. This structuring element will merge caracters that have small gaps in the vertical direction. For instance, dilating the image with this kernel will merge the 2 components formed by the letter *i*.

Lastly, we merge the horizontal and vertical representations using a pixel-wise AND operation. The result, after merging, is depicted in Figure 8. We can see that the combination of horizontal and vertical shapes resulted in clean, well-defined blocks of words. The next step is to find the bounding boxes surrounding each connected component.

Again, we use the OpenCV *connectedComponentsWithStats()* function to find the connected components and their bounding boxes. For each bounding box, we can compute

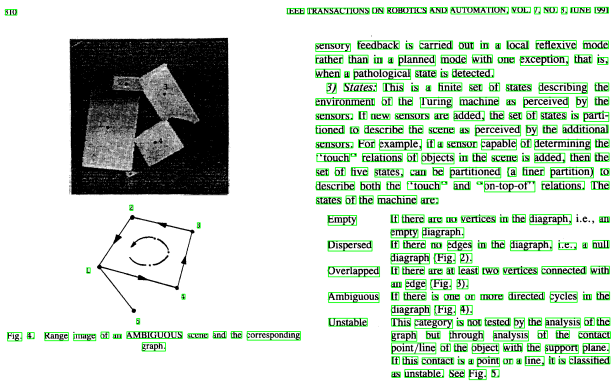


Figure 9. The algorithm was able to count 244 words. Each word is surrounded with a bounding box.

the pixel ratio and the white to black transitions. Finally, we can hand-design heuristics to discriminate a bounding box as text or non-text. The final result can be seen in Figure 9.

Considering numbers as words, and discarding special characters, the text has approximately 241 words. The algorithm was able to detect a total of 244 words, out of 316 possible candidates (number of initial bounding boxes). In summary, the algorithm captured most words in the text. It also classified numbers as words, which is reasonable based on the set of features we used for classification.

3. Discussions

It is important to note that although this method seems robust, it is not general. The shape of the structuring elements does not generalize to other document text. As a result, the method we used we classify components as text or non-text does not generalize either.

If we only use simple heuristics, the proposed solution will probably output different (non-accurate) results for different images. That is due to the chaotic nature of printed documents. Although they seem well behaved in terms of structure, the structuring elements are sensitive to characteristics like font-size, line spacing, and font-type.

A better way of achieving a generalizable set of rules would be to add a learnable component to the system. One approach is to learn unsupervised regions where similar objects are clustered together based on intrinsic characteristics. In other words, we could use a clustering algorithm such as K-Means using the image patches of the components as input.

This way, K-Means would learn to aggregate patches of text together (because they are similar). In the same way, it would learn different clusters for image patches that are not similar to text. Nevertheless, we still would be bounded by the capacity of finding the connected components.

References

[1] Oliphant Travis E. A guide to numpy, 2006. [Online; accessed 15/04/2018]. 1

[2] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000. 1

[3] John D. Hunter. Matplotlib: A 2d graphics environment, 2007. [Online; accessed 15/04/2018]. 1