Combining Morphological and Histogram based Text Line Segmentation in the OCR Context

Pit Schneider[1]

[1]National Library of Luxembourg, Luxembourg

Corresponding author: Pit Schneider , `pit.schneider@bnl.etat.lu`

Abstract

Text line segmentation is one of the pre-stages of modern optical character recognition systems. The algorithmic approach proposed by this paper has been designed for this exact purpose. Its main characteristic is the combination of two different techniques, morphological image operations and horizontal histogram projections. The method was developed to be applied on a historic data collection that commonly features quality issues, such as degraded paper, blurred text, or curved text lines. For that reason, the segmenter in question could be of particular interest for cultural institutions, such as libraries, archives, museums, ..., that want access to robust line bounding boxes for a given historic document. Because of the promising segmentation results that are joined by low computational cost, the algorithm was incorporated into the OCR pipeline of the National Library of Luxembourg, in the context of the initiative of reprocessing their historic newspaper collection. The general contribution of this paper is to outline the approach and to evaluate the gains in terms of accuracy and speed, comparing it to the segmentation algorithm bundled with the used open source OCR software.

Keywords

text line segmentation; morphology; histogram; optical character recognition; historic newspapers

I CONTEXT

While adapting modern open source OCR software to reprocess a large collection of historic newspapers, ranging from 1841 to 1954, the National Library of Luxembourg (BnL) explored ways to segment the newspaper scans into individual text lines. Pursuing this goal, a method was developed that integrates into a larger OCR pipeline by sitting just in between the binarization and font recognition processes. The used open source "kraken" software by Kiessling [2019], forked from the OCRopus OCR System by Breuel [2008], that implements the character recognition step of the pipeline, already comes equipped with a segmentation method. This algorithm, in the following referred to as BENCH, essentially served as a benchmark in the context of the development of an own solution, designed for the precise needs of BnL.

The motivation for the initiative was fourfold:

- Due to the large volume of around 6 million newspaper articles, building a computationally efficient segmenter was the main priority and source of motivation for the project. Next to running faster than BENCH, the aim was for the method to only take up a fraction

of the time needed for the OCR pipeline's character recognition functionality.

- Given the historic data target, another goal was to be able to deal with the common properties of the data, such as degraded paper quality, blurred text, presence of noise or curved lines. Problems, as seen with example text block image depicted in Figure 1, appear on a frequent basis in the dataset.
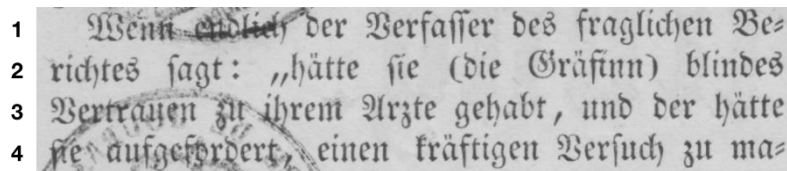

Figure 1: Example text block of historic newspaper collection, featuring four text lines and the potential issues of background noise (stamp) and border presence (right).

- Furthermore, BENCH is advertised as a page segmenter, including layout analysis. The algorithm can however be passed a parameter, indicating that a single single column of text (i.e. zero column seperators) is to be expected. Naturally this was the chosen configuration for comparison purposes. Since the single column property is always valid for the presegmented text blocks in the BnL dataset, a solution, purely designed for line detection within a single block, was favored. The overall aim was to develop a simple algorithm that only returns a set of line bounding boxes and does not incorporate additional features, such as general text direction (left-to-right, top-to-bottom, ...) detection, for example.

- Lastly, developing, understanding and controlling a proprietary solution, is a welcoming advantage. Approaching the problem this way supports advantages in future maintenance and supports the generation of in-house technical competencies.

Having laid out the context of this work, the remainder of this paper is structured in two sections, followed by concluding statements. The first one formally defines the algorithm, while reusing the same example input to illustrate intermediate results. The subsequent section evaluates the method and draws the comparison to BENCH.

II   ALGORITHM

Given that the interplay of morphology and histogram projections can lead to very encouraging results, as observed by Dos Santos et al. [2009], the combination was used as a foundation to build on. The nature of combined techniques is also the reason why the proposed segmentation method is referred to as COMBISEG. In the following, different subsections will cover the algorithm's input requirements, the functioning of the segmentation itself and the resulting output.

2.1   Input Assumption

To define the scope of the approach and consequently clarify the input requirements, it is sufficient to state that the algorithm expects an input image fulfilling two properties:

1. The input image, in the following denoted I^b, must be presented in a binarized form, composed of ones (text/white) and zeros (background/black) only. This paper represents I^b as a 2D array, with the height $h(I^b)$ seen as the first, and the width $w(I^b)$ as the second dimension. It follows that the highest indices are $h(I^b)-1$ and $w(I^b)-1$ respectively.

2. COMBISEG is not designed to perform layout analysis (e.g. identifying text blocks within a full newspaper page). That's why the second requirement is that I^b is a block of text, solely containing horizontal and typed text lines (not tested for handwriting). In the example of the BnL use case, the employed dataset was already presegmented into blocks of text, mostly representing paragraphs or small articles.
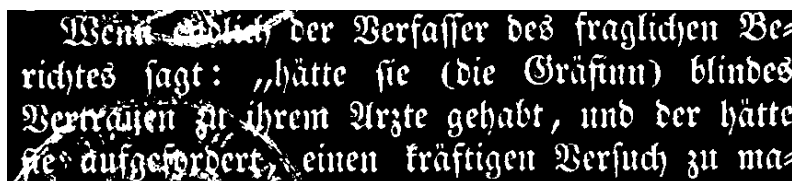

Figure 2: Valid example COMBISEG input.

2.2 Segmentation

Given the correct input, the actual algorithm can be be summed up as a sequence of three consecutive steps.

1. *Morphology*
   First, a set of morphological image operations is applied on I^b, in order to create a new processed version, denoted I^p.

2. *Components*
   Next, I^p is used to search for connected components, with each component potentially already representing a single text line.

3. *Histogram*
   Finally, a horizontal histogram projection is in turn created for each component, being the basis for possible additional text line splits.

The algorithm is also subject to a set of parameters, allowing slight tuning and referred to as $params = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8\}$.

Having laid out the high level plan of COMBISEG, this article will continue with explanations regarding the precise working of the three steps involved.

2.2.1 Morphology

To begin with, a set of sequential image operations is applied to obtain a processed image version, with the intention that every connected pixel component represents a text line. For explanation purposes, this paper will, in the following, make use of the common image operations

$opening\ (\circ)$, $dilation\ (\oplus)$, $addition\ (+)$, $subtraction\ (-)$ and $inversion\ (^{-1})$. To provide an example,

$$I^p \leftarrow I^b \circ E_{w \times h} \tag{1}$$

states that image I^p is obtained by opening image I^b using structuring element $E_{w \times h}$, which is filled with ones only, for width w and height h. Image operations are also considered to return a modified copy, rather than changing the image that's being operated on.

Algorithm II.1 Transforms binary I^b into processed I^p, featuring connected components as potential text lines.

---

1   **morph**(I^b)
2             $I^p \leftarrow I^b - \left((I^b \circ E_{1 \times p_1}) + (I^b \circ E_{p_1 \times 1}) \right)$
3             $I^p \leftarrow (I^p \oplus E_{p_2 \times 1})^{-1}$
4             $S \leftarrow \left(I^p - (I^p \circ E_{1 \times p_3}) \circ E_{p_4 \times 1} \right) \oplus E_{p_5 \times 1}$
5   return $(I^p + S)^{-1}$

Given the pseudocode seen in Algorithm II.1, the line statements can be explained as follows:

• Line 2:

Considered to be a preprocessing operation. The first statement of Algorithm II.1 tries to remove long vertical and horizontal elements (borders, lines, frames, ...) from I^b through opening, followed by subtraction. Hence, every element has to have minimum height/width of p_1 to be considered. The main motivation here is to deal with the potentially cropped nature of the input images. For instance, in the concrete example of I^b having been cropped out of a newspaper page, imprecise croppings and the inclusion of neighbouring article separator lines can be the cause for segmentation algorithms to falsely combine multiple successive text lines.

Figure 3: Intermediate result after $morph$ preprocessing step, almost entirely removing the border line at the very right.

• Line 3:

It follows that all text areas in the preprocessed version I^p are horizontally dilated using structuring element $E_{p_2 \times 1}$, in an attempt to concatenate consecutive characters of the same text line (connected components creation). The obtained result is subsequently inverted (white background) to allow incoming background modifications.
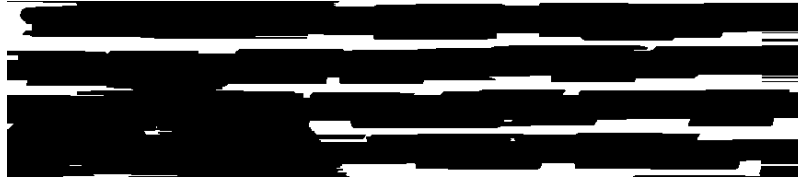
Figure 4: Horizontally dilated and inverted version.

- Line 4:

The next statement starts by removing vertical elements from I^p using an opening operation (protective step). This helps to prepare the background of the image (as seen on Figure 4), in a way that it should prevent the next step (horizontal dilation) to cause damage to the text areas. This next step consist of searching for horizontal background separators. Once found, they are dilated horizontally to increase their width. This should break unwanted connections between successive text lines (e.g. second and third text line in Figure 4), that might have been created by the very first dilation. The protective first step thus attempts to avoid that background separators are detected in front or after text lines, that don't extend over the entire image width, and consequently cut into the text areas. The result of the entire statement is stored in S, representing the separators between text line areas.

Figure 5: Representation of text area separators S that have been reinforced compared to Figure 4.

- Line 5:

Adding S to the inverted image version translates to the subtraction from the regular one. This explains the process of the last algorithm statement. As mentioned in the previous paragraph, the separators attempt to break unwanted text area connections. The final I^p of the example input image, as returned by Algorithm II.1, can be seen with Figure 6. Comparing with Figure 4, the addition of S ends up being the catalyst for the separation of text lines two and three.
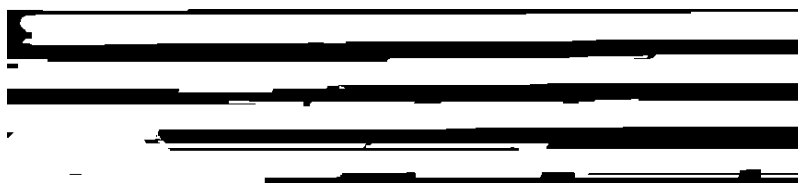

Figure 6: Processed image I^p originating from binary image I^b.

2.2.2 Components

Using I^p, all connected white pixel components are identified. For this purpose, the 4-way-connectivity is used (i.e. diagonal pixels are not considered as being connected). Next, all

component bounding boxes, using coordinates within I^p, are determined and stored in a list of no particular order.

Let n denote the number of found components. To determine bounding box_i of the ith component, also let X_i and Y_i represent the set of all x-coordinates and y-coordinates in I^p, of all pixels belonging to the ith component. It is defined that

$$box_i = \big[min(X_i), min(Y_i), max(X_i), max(Y_i)\big]. \tag{2}$$

Finally, $boxes$ is used to store a first version of the bounding boxes that the algorithm will ultimately return. At the same time, bounding boxes that don't have minimum height p_6 are discarded, in a way that

$$boxes = \big\{box_i \mid box_i[3] - box_i[1] \geq p_6\big\}, \text{ with } 1 \leq i \leq n, \tag{3}$$

if $|boxes| = 0$:
$$boxes = \big\{[0, 0, w(I)-1, h(I)-1]\big\}.$$

Using the process described in this subsection, a $comp$ function is defined as

$$comp(I^p) \rightarrow boxes. \tag{4}$$

Figure 7: Visualization of $n=3$ connected components with minimum height p_6.

2.2.3 Histogram

The final but essential step of COMBISEG is the analysis of horizontal histogram projections, one created for every bounding box stored in $boxes$. The global projection of I^b is denoted as H^*, an array storing the number of white pixels in every row of the image, s.t.

$$H^*[j] = \sum_{k=0}^{w(I)-1} I^b[j][k], \text{ with } 0 \leq j \leq h(I)-1. \tag{5}$$

Figure 8: Representation of global projection H^* of example image.

Through the selection of the subset values $H^*\big[box_i[1]..box_i[3]\big]$ (i.e. all values that fall in the vertical component range) and their subsequent reordering, a sub-projection H_i is created for

every ith component from 1 to n. This sub-projection is considered to be a dictionary, ordered by decreasing histogram values, while the y-coordinates are figuring as dictionary keys. This process is in the following referred to as

$$subproj(H^*, box_i) \rightarrow H_i. \tag{6}$$

The sub-projection is then passed to the $analysis$ procedure (Algorithm II.2), an algorithm proposed by Ptak et al. [2017], that analyzes the locations of peaks in the histogram data. Next, based on the identified peaks, the procedure suggests division y-coordinates to split the underlying connected component bounding box.

This is done by identifying peaks using a relative threshold α_t, having a value proportioned to the height of the highest location of the potential peak, which is equal to the height h at $coord$. The set C keeps track of already analyzed coordinates. Should the new range of peak coordinates R be disjoined with C, a new peak has been identified and its delimiters p_{start} and p_{end} are added to the peaks set P.

Algorithm II.2 Analyzes sub-projection and returns possible y divisor coordinates.

1	**analysis**$(H_i, t = p_7)$
2	$\quad C \leftarrow \emptyset$
3	$\quad P \leftarrow \emptyset$
4	$\quad$ for $coord$ in H_i
5	$\quad\quad h \leftarrow H_i[coord]$
6	$\quad\quad$ if $h < 0.1 \times max(H_i)$ break
7	$\quad\quad$ if $coord \notin C$
8	$\quad\quad\quad \alpha_t \leftarrow t \times h$
9	$\quad\quad\quad p_{start}, p_{end} \leftarrow bounds(H_i, coord, \alpha_t)$
10	$\quad\quad\quad R \leftarrow [p_{start}..p_{end}]$
11	$\quad\quad\quad$ if $C \cap R = \emptyset$
12	$\quad\quad\quad\quad P.append \leftarrow p_{start}, p_{end}$
13	$\quad\quad\quad C \leftarrow C \cup R$
14	$\quad$ return $valleys(H_i, P)$

The $analysis$ procedure in turn calls two different helper functions:

- $bounds(H_i, coord, \alpha_t)$

  Accepts an ordered dictionary sub-projection H_i, a y-coordinate $coord$ and the height threshold α_t as input. The function computes and returns the delimiting coordinates p_{start} and p_{end} of the histogram peak located at $coord$. This is done by iteratively increasing/decreasing $coord$ by one, depending on the bound that should be discovered. A delimiting coordinate has been found, once the value of H_i, for the given new coordinate, falls below the relative threshold α_t. It follows that

  $$H_i[p_{start}] \geq \alpha_t \ \land \ H_i[p_{start}-1] < \alpha_t \ \land \ H_i[p_{end}] \geq \alpha_t \ \land \ H_i[p_{end}+1] < \alpha_t \tag{7}$$
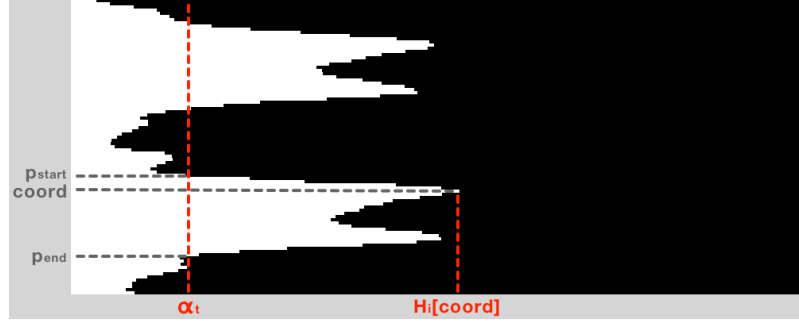
  is satisfied.

Figure 9: Visualization of *bounds* applied to sub-projection of third component (text lines three and four).

- $valleys(H_i, P)$

  The same sub-projection, together with all the computed peak delimiting coordinates P is passed here. Once the set P is ordered increasingly, the procedure computes splitting y-coordinates, representing valley locations in the histogram data. This is done by first discarding the first (lowest) and last (highest) value in P. Next, the range of coordinates between every consecutive pair of the remaining values ($[p_{end}..p_{start}]$ ranges) is considered. Finally, the coordinate of every range, that reveals the lowest histogram value in H_i, is appended to the returning $splits$ list. In case of $|P| < 4$, no splitting coordinate is returned, since the projection features at most a single peak (i.e. connected component represents a single text line).

Now that it has been established how $splits$ can be computed for every connected component found in I^p, it remains to be defined how the splitting coordinates are to be used.

Algorithm II.3 Potentially splits box_i into multiple boxes using the $splits$ coordinates.

---

1   **split**$(splits, box_i)$
2         $boxes^u \leftarrow \emptyset$
3         $splits \leftarrow splits \cup box_i[3]$
4         $last \leftarrow box_i[1]$
5   if $|splits| > 0$
6   for s in $splits$
7   if $s - last \geq p_6$
8                   $box \leftarrow [box_i[0], last, box_i[2], s]$
9                   $boxes^u.append \leftarrow box$
10              $last \leftarrow s$
11   else
12             $boxes^u.append \leftarrow box_i$
13   return $boxes^u$

Algorithm II.3 iterates through the $splits$ list and creates new bounding boxes by splitting at every coordinate s in case the resulting box respects the minimum height of p_6.

Algorithm II.4 Transforms initial $boxes$ into updated $boxes^u$ using histogram information of I^b.

---

1   **hist**$(I^b, boxes)$
2          $boxes^u \leftarrow \emptyset$
3   for box_i in $boxes$
4              $H_i \leftarrow subproj(H, box_i)$
5              $splits \leftarrow analysis(H_i)$
6              $boxes^u.append \leftarrow split(splits, box_i)$
7   return $boxes^u$

Bringing everything together, an updated set $boxes^u$ is computed as indicated in Algorithm II.4.

2.3 Output

Algorithm II.5 ultimately provides the interactions between all three major steps involved in COMBISEG.

Algorithm II.5 The COMBISEG high level algorithm.

---

1   **CombiSeg**(I^b)
2          $I^p \leftarrow morph(I^b)$
3          $boxes \leftarrow comp(I^p)$
4          $boxes^u \leftarrow hist(I^b, boxes)$
5   return $adjust(boxes^u)$

What remains to be clarified is the call to $adjust$ in Line 5. As a postprocessing procedure, the bounding boxes are modified one last time, in a way that:

- They are sorted by increasing $box_i[1]$.

- Their height is increased using parameter p_8, s.t. updated box_i equals

$$box_i = \big[box_i[0], box_i[1] - p_8, box_i[2], box_i[3] + p_8 \big]. \tag{8}$$

  This increase is justified by the fact that the background dilation in $morph$ can be the cause of a slight height reduction, which should be compensated in $adjust$. In addition, it can be that bounding boxes with a greater amount of padding are preferred.

- Lastly, $adjust$ merges all pairs of bounding boxes where box_i completely contains box_j, in a way that

$$(box_i[c] \leq box_j[c] \; \forall c \in [0..1]) \; \wedge \; (box_i[c] \geq box_j[c] \; \forall c \in [2..3]), \quad \text{with} \quad i \neq j. \tag{9}$$

Optionally, $adjust$ provides the occasion to add custom rules defining the merging of vertically overlapping bounding boxes (see Subsection 3.2).

The result of the application of COMBISEG can be seen with Figure 10. In the case of the example input, the blue component in Figure 7 was the basis for a successful detection (by *analysis*) of two different peaks, which have been *split* and turned into individual bounding boxes (overcoming the potential background stamp issue).
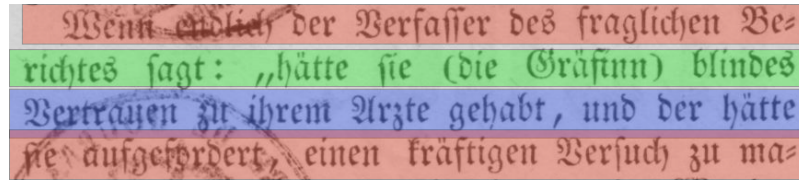

Figure 10: Visualization of the four resulting output bounding boxes of COMBISEG.

2.4 Parameters

In order to adapt to a specific target dataset and its proprietary characteristics, the set of *params* can be tuned accordingly. The respective parameters, together with their usage description, are listed below:

- p_1: Preprocessing element height/width.
- p_2: First (text) dilation width.
- p_3: Background protection element height.
- p_4: Background separator width.
- p_5: Second (background) dilation width.
- p_6: Minimum text line height.
- p_7: Relative peak height threshold.
- p_8: Postprocessing height adjustment.

2.4.1 Value Determination

The parameter values, experimentally leading to the best results on a BnL test set (more details follow in Section III), are given by

$$params = \{100, 90, 25, 35, 330, 14, 0.3, 5\}. \tag{10}$$

To provide some guidance on the approach of identifying (near) optimum values, the following three-step course of action is proposed:

- Largely independent of every other parameter in the set, are p_1, p_6 and p_8. Their values are generally determined by "preference choices". While p_1 should be considerably (depending on the aggressiveness of the desired preprocessing) greater than a regular text line height, p_6 should obviously be smaller to solely filter out noise.

- Next, there are the morphology related parameters p_2, p_3, p_4 and p_5. A similar order of magnitude, with respect to the ratio compared to the the average text line height (42.9 pixels for the BnL test set), should be chosen as a starting point in every case. Afterwards, a small amount of discrete values should be brute forced (e.g. deltas of $-10, 0$ and $+10$, leading to $3^4 = 81$ individual runs) by applying all combinations on a relatively small test

set. It is of course essential to evaluate COMBISEG on the output of $morph$ and $comp$ only, discarding the subsequent $hist$ component.

- Lastly, p_7 should be determined independently for a given set of fixed morphology values. Hence, this final parameter can be learned in a supervised way through the overall performance of COMBISEG.

III EXPERIMENTAL RESULTS

For the purpose of evaluating the performance of COMBISEG, a BnL ground truth dataset is used and referred to as being a set of tuples $(I^b, gt) \in GT$. It contains 5,958 text block images cropped from historic newspapers with ground truth bounding boxes originating from human annotators. The number of text lines per sample ranges from 2 to 143 for a total of 11,4625 lines. All images are available in the TIFF format with a 300 PPI density.

3.1 Evaluation Method

In order to run COMBISEG against GT and compare the output to every set of bounding boxes gt, an evaluation method is required. For that purpose, a $loss$ function is used to return the number of unmatched text lines for a given test image. Next to deducting the number of correctly $matched$ lines from cardinality $|gt|$, the function also punishes the detection of superfluous text lines, while enforcing an overall upper bound of $|gt|$, in a way that

$$loss(boxes^u, gt) \;=\; min\big(|gt|, |gt| - |matched| + max(0, |boxes^u| - |gt|)\big), \tag{11}$$

where

$$matched = \Big\{ a \in gt \;\Big|\; \exists b \in boxes^u \; s.t. \; abs\Big(\frac{a[1] + a[3]}{2} - \frac{b[1] + b[3]}{2}\Big) \leq \theta \Big\}. \tag{12}$$

Commenting on (12), a trivial definition that compares the middle y-coordinates of bounding boxes, is established. This way the $loss$ function represents the correct identification of individual text lines (main challenge in the historic dataset), rather than focusing on perfect bounding box dimensions (e.g. intersection with ground truth bounding box pixels).

The threshold value θ is set to equal 14.3, representing a third of the average bounding box height in GT of 42.9 pixels. Hence, a matched bounding box still has considerable vertical shifting possibilities, making the $loss$ function even more sensitive to the identification of text lines.

Using (11), the accuracy of COMBISEG can be compared to BENCH, by computing

$$acc(algo) = 1 - \frac{\sum_{(I^b, gt) \in GT} loss\big(algo(I^b), gt\big)}{\sum_{(I^b, gt) \in GT} |gt|} \tag{13}$$

for any algorithm $algo$.

3.2 Postprocessing

The conducted experiments required postprocessing to make both algorithms more comparable. This is backed by the fact that BENCH tends to pursue a higher degree of fragmentation. Multiple bounding boxes are generally generated for text that is horizontally aligned correctly, but features particular large spaces between words. Since BnL considers those cases to represent one single text line, custom rules had to be introduced to merge bounding boxes based on their vertical positions. Those rules are equally applied to the output of both, BENCH and COMBISEG (in the context of the call to $adjust$).

Let $box_i[1] \leq box_j[1]$, with $i \neq j$, denote the starting y-coordinates of two successive bounding boxes in the list of all output boxes, ordered based on this value. A vertical overlap between box_i and box_j is defined as

$$o = max(0, box_i[3] - box_j[1]). \tag{14}$$

A merger to a combined single bounding box is performed, in case there is

$$\left(\frac{o}{box_i[3] - box_i[1]} > 0.75 \right) \vee \left(\frac{o}{box_j[3] - box_j[1]} > 0.75 \right) \vee \left(\frac{o}{box_j[3] - box_i[1]} > 0.5 \right). \tag{15}$$

Commenting on (15), boxes are merged in case o represents more than 75% of the height of one of the single boxes, or more than 50% of the combined height of both boxes.

3.3 Results

By applying the constructed evaluation method of Section 3.1, together with the just discussed postprocessing rules, the experimental results based on GT are

$$acc(\text{COMBISEG}) > acc(\text{BENCH})$$
$$\implies 1 - \frac{890}{11,4625} > 1 - \frac{2066}{11,4625}$$
$$\implies 0.992 > 0.982. \tag{16}$$

Although observing improved accuracy is welcoming, the main objective of BnL was to improve computational efficiency with COMBISEG. Running both algorithms for GT on the exact same hardware, delivered very promising results related to the average processing time, in a way that

$$pt(\text{COMBISEG}) < pt(\text{BENCH})$$
$$\implies \frac{101,778}{5,958} < \frac{4,583,208}{5,958}$$
$$\implies 17.08 < 769.25, \tag{17}$$

using trivial formula

$$pt(algo) = \frac{total\ processing\ time\ (ms)}{number\ of\ processed\ images}. \tag{18}$$

It should be stated that, for the purpose of this experiment, COMBISEG was implemented based on methods from the popular OpenCV library (Bradski [2000]). On the other hand, the implementation of BENCH was run as published with version $2.0.8$ of the software.

IV CONCLUSION

In summary, the proposed segmentation method COMBISEG is a perfect fit for the BnL use case of segmenting newspaper articles into individual text lines. The algorithm performs really well in terms of both, accuracy and speed, thus having a positive effect on the overall OCR initiative. On top of that, it has the advantage of not requiring a separate training phase, as it is the case for model based approaches. Its overall robustness originates from the combination of two techniques, commonly used in the literature. Morphological operations and horizontal histogram projections are combined, in a way that they complete each other and cancel out their weaknesses.

- Generating connected components through dilation has the advantage of obtaining rather precise coordinate information. However, this technique is prone to be fooled by background noise or closely adjacent lines, leading to unwanted connections and potentially major segmentation mistakes.

- Histogram projections, on the other hand, are less sensitive to anomalies and mostly overcome them by considering the integrity of image rows. Nonetheless, evaluating the histogram data and finding the correct bounding box through peak/valley detection, is not a trivial job. This is where the preliminary morphology work is able to help.

Overall, COMBIGSEG marks a promising segmentation algorithm candidate, especially for OCR purposes in relation with relatively challenging historic datasets.

References

G. Bradski. The opencv library. *Dr. Dobb's Journal of Software Tools*, 2000.

T. Breuel. The ocropus open source ocr system. 2008.

R. Dos Santos, G. Clemente, T. Ren, and G. Calvalcanti. Text line segmentation based on morphology and histogram projection. 2009.

B. Kiessling. Kraken - a universal text recognizer for the humanities. 2019.

R. Ptak, B. Ygadlo, and O. Unold. Projection–based text line segmentation with a variable threshold. *International Journal of Applied Mathematics and Computer Science*, 27, 2017.