Automatic Localization and Correction of Line Segmentation Errors

Anand Mishra* Naveen Sankaran Viresh Ranjan C. V. Jawahar

Center for Visual Information Technology, IIIT Hyderabad, India
http://cvit.iiit.ac.in/

ABSTRACT

Text line segmentation is a basic step in any OCR system. Its failure deteriorates the performance of OCR engines. This is especially true for the Indian languages due to the nature of scripts. Many segmentation algorithms are proposed in literature. Often these algorithms fail to adapt dynamically to a given page and thus tend to yield poor segmentation for some specific regions or some specific pages. In this work we design a text line segmentation post processor which automatically localizes and corrects the segmentation errors. The proposed segmentation post processor, which works in a "learning by examples" framework, is not only independent to segmentation algorithms but also robust to the diversity of scanned pages.

We show over 5% improvement in text line segmentation on a large dataset of scanned pages for multiple Indian languages.

1. INTRODUCTION

The failure in text line segmentation profoundly affects the overall accuracy of an OCR engine. Line segmentation algorithms are one of the widely studied and evolving topics in document image analysis literature [1, 9, 15, 6]. Most of these segmentation algorithms perform satisfactorily well, but tend to fail in some specific region or for some specific pages. The main cause of such failures is that these algorithms are heavily dependent on the parameters and thus fail to adapt to a given page dynamically. Moreover, the interest of the document image analysis community in this area is also obvious from the consistent appearance of page segmentation work (see [2], [3], [4]) in competitions at ICDAR.

Many segmentation algorithms have been proposed in literature. These algorithms can be grouped into three categories: top-down [13], bottom-up [16, 5] or hybrid algorithms [1]. Nevertheless, the complexity and variations in the document images make the task of segmenting a given document page into lines still challenging. The exhaustive experiments on scanned document of a large collection of Indian language dataset are conducted in [10, 12]. These experiments indicate that the well known algorithms tend to perform poor for these languages. This is mainly due to characteristics of the scripts.

Conceptually, any line segmentation algorithm tries to optimize an objective function such that the inter line variance and the variance of the line heights is minimized. The popular segmentation algorithms [1, 5, 13, 16] in literature do it in a greedy way. A heuristic refinement of segmentation is also an integral part of many of the good implementation of these algorithms. Such algorithms do not take an advantage of training examples to learn the parameters. We take a different path and propose a methodological way of solving this problem by designing a segmentation post-processor which automatically localizes and corrects the text line segmentation errors. The error localization and correction is driven by training examples.

Extending the work of [12] on localizing segmentation errors, we design a post-processor which automatically localizes and corrects the errors. For this we formulate the problem of locating line segmentation errors as a multi-class classification problem, where each segmented line is classified into one of the five classes *i.e.*, correct, under-segmented, over-segmented, false alarm or missing dangling modifier. For this we use a small set of document images as a training data and assume availability of ground truth for this. We, then compute some line level features (described in Section 4.2) for every segmented line in the training data and a given page. Once these features are computed, each line in a given page is classified into either one of the error type or correct. Once segmentation errors are localized, we correct those based on the error type and confidence of error using our segmentation correction algorithm. The segmentation post-processor transforms the segmented line such that its probability of becoming correct increases.

The proposed system localizes the segmentation errors based on learning from the examples of a training set, thus the method is robust and applicable to a wide variety of pages. We experimentally show improvement in text line segmentation by applying the post-processor to the segmentation output of a projection profile based algorithm.

The remainder of the paper is organized as follows. In Section 2 we discuss the related work of the problem. In Section 4, we describe the proposed segmentation post-processor *i.e.* automatic localization and correction of line segmentation errors. Section 5 elaborates experiments and results.

*anand.mishra@research.iiit.ac.in

DAR '12, December 16, 2012, Mumbai, India

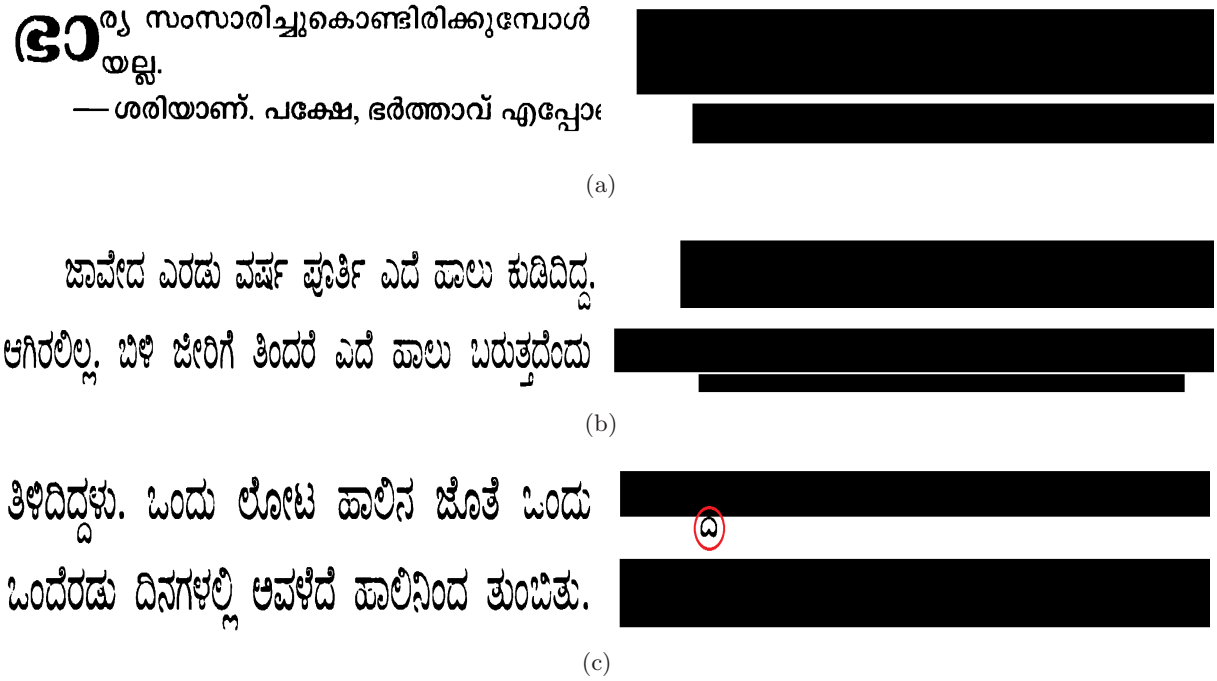


Figure 1: Typical segmentation errors in Indian scripts as discussed in [12] (a) Two lines are merged into one line (under-segmentation) (b) One line is spilt into two lines (over segmentation) (c) A dangling modifier shown in a small red circle is missed (missing component). Missing component errors especially occur in many Indian languages.

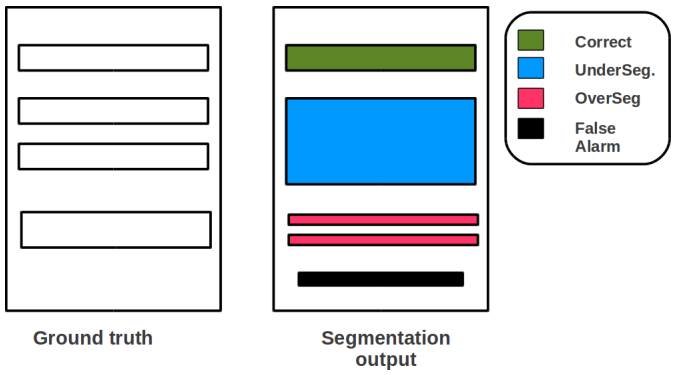


Figure 2: Various Segmentation Errors with respect to ground truth. Here colour green, blue, red and black represent correct segmentation, under-segmentation, over-segmentation and false alarm respectively. (Best Viewed in Colour).

We finally conclude our work in Section 6.

2. RELATED WORK

Many segmentation algorithms have been proposed in literature. The popular ones are Recursive XY cut [13], White-space analysis [5], Docstrum [14], Voronoi diagram based [8], and RLSA [16]. Description of these algorithms is not in the scope of this work. However, readers are encouraged to refer [15] for this. There are many ways to classify these algorithms, the most popular being as top-down and bottom-up. In the first category, page is recursively segmented until certain termination conditions are met. While the bottom up methods group segments and form words, lines and blocks. There are also methods which combine these two.

Most of these segmentation algorithms suffer from some or other line segmentation errors. Some typical example of line segmentation errors are shown in Figure 2. These errors often occur due to the failure of segmentation algorithms to adapt the parameters locally. Segmentation algorithms could have a global parameter, or even have a local parameter. However, defining all the possible failure cases and thereafter adjusting the parameters make the algorithm highly heuristics. Such heuristic algorithms (and their implementations) often become very brittle. (*i.e.*, fails in unpredictable manner in unknown situations). As a result, to make the algorithm more predictable, researchers often design principled methods. These principled methods may show larger errors than heuristic methods on selected pages; but they become more predictable and analyzable.

In this work, we want to go one more step forward. We want to *automatically* detect the failures of the segmentation algorithms (*i.e.*, without a ground truth) and then rectify the errors by *post-processing* locally.

Modern computer vision community has shown interest in the similar line, where image segmentation algorithms are evaluated automatically *i.e.* without using ground truth.[17] provides a detail survey of automatic (unsupervised) evaluation of image segmentation algorithms. In unsupervised evaluation of segmentation, availability of the ground truth is not assumed. Rather a set of features are computed from the segmented image and based on these features perfor-

mance of image segmentation algorithms are measured. We are highly inspired by such methods and in this work we go one step further and not only automatically localize the line segmentation errors but also correct them.

3. SEGMENTATION ERRORS

The text line segmentation errors can be can be defined in a set theocratic notation as in [15]. For the sake of completeness, we summarize these definitions here. Let S and G be the set of lines denoting segmentation output and ground truth respectively. Then we can define segmentation errors as follows.

Correct.

A Line $B \in S$ is said to be correct if there exists a unique line $A \in G$ such that $A \cap B$ is significant.

Over-segmented.

A line $B \in S$ is said to be over-segmented if there exist at-least one more line $B^{'} \in S$ and $A \in G$ such that both $A \cap B$ and $A \cap B^{'}$ are significant.

Under-segmented.

A line $B \in S$ is said be under-segmented if there exist multiple lines A's in G such that $A \cap B$ is significant.

Missing component.

A line $B \in S$ is said to be missing component if there exists a unique line $A \in G$ such that $A \cap B$ is not significant, *i.e.*, by calling line A as missing component we mean that line A has missed some dangling modifier either above or below the line.

False alarm.

A line $B \in S$ is said to be a false alarm if there does not exist any line $A \in G$ such that $A \cap B \neq \phi$.

Figure 2 explains these errors pictorially. Here colour green, blue, red and black represent correct segmentation, under-segmentation, over-segmentation and false alarm respectively.

4. THE SEGMENTATION POST-PROCESSOR

In this section, we describe the basic ideas related to error localization and correction. It may be noted that we do not propose any new algorithm for page segmentation. Rather, we redefine the popular segmentation schemes by introducing an automatic error detection and correction module.

4.1 Overview

We are interested in two modules: (i) A module which can detect errors automatically, without a ground truth. This module is designed as a classifier which is trained from a set of examples. To be precise, our implementation classifies the regions as erroneous or not, and also label with the type of error (eg. over-segmentation). (ii) The second module looks at the erroneous regions more carefully and refine the segmentation to minimize the errors. We call these two stages as automatic error localization and correction. In Section 5, we show that our detection module can detect close to 85% of the errors present in many of the Indian scripts. After that more than 60% of these errors are removed by the second module. Before we describe the details of how these two modules are designed, we give a simple intuitive explanation.

We observe that (1) most of the characters in a page are of same size, font and style, (2) line spacing within the documents are mostly same, (3) page is formatted uniformly within a book, (4) two nearby lines in a document is mostly of same height. Based on these simple intuitions, we compute a set of features (which we explain soon) and use training data to learn segmentation errors. This helps us to automatically localize text line segmentation errors. To make the localization robust, we use SVM classifier. Moreover, SVM classifier is also useful as it gives confidence score of classification, which we use to design a powerful post-processor. Once the error are localized, we apply a set of transformations to the incorrect segmented lines such that the probability of its becoming correct increases. The block diagram in Figure 3 presents the concept of the proposed segmentation post-processor.

In this section, first we explain the strategy for automatic localization of text line segmentation errors and then elaborate algorithm for the segmentation correction.

4.2 Automatic Error Localization

We localize the text line segmentation errors in a supervised learning framework. For this we assume the availability of the ground truth for few pages and compute a set of features for every segmented line.

The set of features which we compute for each line L_i in order to classify it as correct, over-segmented, under-segmented, false alarm or missing component are as follows:

F1: Difference in line heights and line gap.

We define this feature as follows:

$$F1 = max\{|LH_{i-1} - LH_i| - LG_{i-1,i}, \\ |LH_i - LH_{i+1}| - LG_{i,i+1}\},$$

where LH_i is the height of line i and $LG_{i-1,i}$ is the gap between lines $i-1$ and i. The intuition behind this feature is two closest line should be of similar height.

F2: Difference in line height and maximum height of connected component.

We use difference of line height and maximum height of connected component in a line as a feature. This helps us to locate under segmentation where line height is far greater than size of maximum connected component.

F3: Maximum area of CCs closest to line.

To define this feature for a line L_i we find out the CCs which are not part of any line and is the closest to line L_i compared to its above or below lines *i.e.*, lines L_{i-1} and L_{i+1}. We compute the area of all such CCs and take maximum area as a feature F3 for line L_i

F4: Maximum area of connected component in a line.

In every line we compute the maximum area of connected component and use it as a feature. Very high and low value of this feature correspond to false alarm.

F5: Minimum of upper and lower line gaps.

We define feature F5 for the line L_i as:

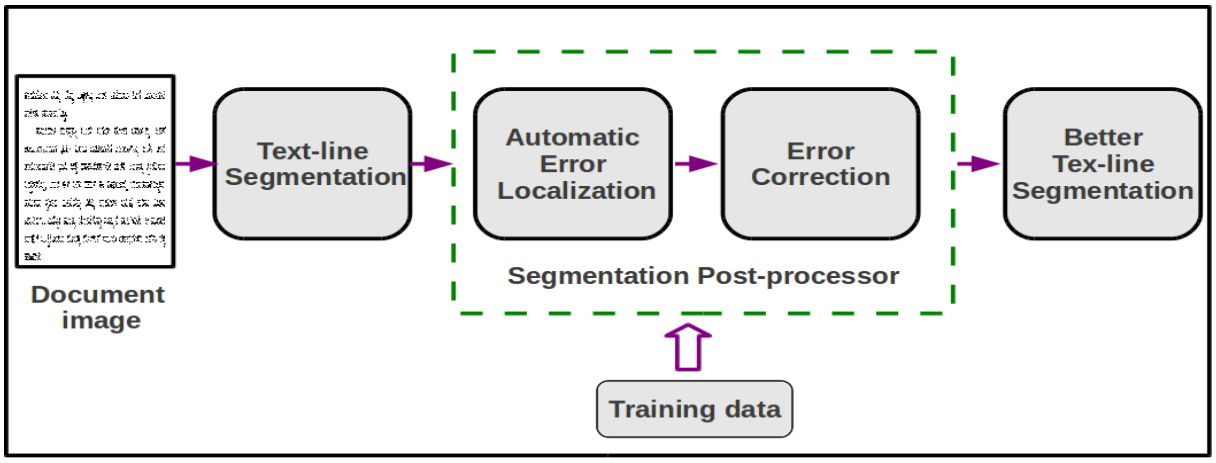


Figure 3: The block diagram explains the functionality of the segmentation post-processor.

$$F5 = min\{LG_{i-1,i}, LG_{i,i+1}\}.$$

This feature helps us to locate over-segmentation where one of the gap $LG_{i-1,i}$ or $LG_{i-1,i}$ is very low.

We compute these features for every segmented line of the training pages and a given test page. We, then learn the associated segmentation errors for every segmented line in training pages. However, for a given test page, we do not use the ground truth and localize the errors using SVM classifier. In other words, we formulate the problem of automatic error localization as a multi-class classification framework where each line is classified (tagged) as correct, over-segmented, under-segmented, false alarm or missing component. More formally, For a segmented page with lines $\{L_1, L_2, ..., L_n\}$ we classify each line as (1) Correct (co) (2) Over-segmented (os) (3) Under-segmented (us) (4) False alarm (fa) and (5) Missing component (mc)

Further, to make the post-processor robust to small errors in classification (*i.e.* automatic localization), we also compute confidence of each classification. In other words, the automatic localization module returns a tuple $\{L_i, E_i, C_i\}$ for every segmented line L_i. Here $E_i = \{co, os, us, fa, mc\}$ denotes the segmentation error type and C_i denotes the confidence of line L_i classified as error E_i.

4.3 Error Correction

The error correction module takes the output of automatic error localization as Input. The error correction can be viewed as a transformation of features of the segmented lines such that the probability of correctness of these segmented lines increases. The transformation of features is driven by following set of rules. If automatic localization classifies line as correct, no action is required. If it classifies line as false alarm with very high classification confidence, the line is deleted. If automatic localization tags line as overs-segmented and the line next to it is also tagged as over-segmented, then these two lines are merged. If automatic localization tags line as missing component with very high classification confidence, then the CCs closet to the line (either on upper or lower part of segmented line) is included in the line. Finally, if automatic localization tags line as under-segmentation with very high classification confidence, we iteratively change the thresholds and re-run the segmentation algorithm so that it produces more than one line. Compute the line level features for new lines and classify using automatic localization module until the re-produced lines are classified as correct with very high confidence. The algorithm 1 summarizes these transformations.

In summary, for a output of a segmentation algorithm, we automatically localize the text line segmentation errors. We then, use the output of automatic error localization for error correction. The error correction module uses the error information and confidence of classification to correct the errors. This finally yields an improved text line segmentation.

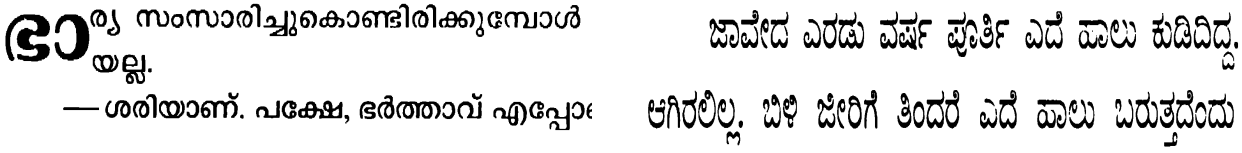
Algorithm 1 Error correction

```
Input: {(L_i, E_i, C_i) : i ∈ {1, 2, ..., n}}
Output: {L_j : j ∈ {1, 2, ..., m}}
for i = 1 to n do
  if E_i = co and C_i is high then
    L_j ← L_i
  else
    if E_i = fa then
      L_j ← φ
    end if
  else
    if E_i = os and L_{i+1} = os then
      L_j ← merge(L_i, L_{i+1})
    end if
  else
    if E_i = mc and C_i is high then
      L_j ← extend(L_i)
    end if
  else
    if E_i = us then
      split L_i into lines L_{x_i} till E_{x_i} = co, ∀i
      L_j ← {L_{x_i}}
    end if
  end if
end for
```

5. EXPERIMENTS AND RESULTS

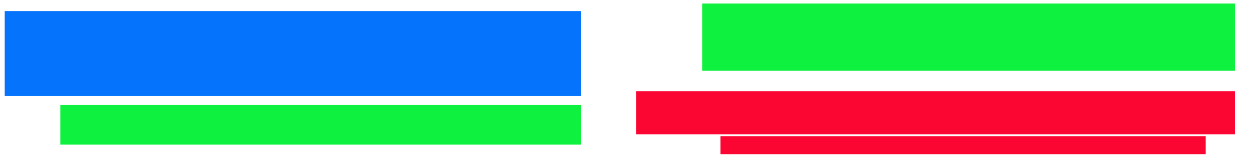
We use a dataset [7] of 8 books with more than 41K text-lines. The script of these books belongs to Indian languages. These books were identified based on experiments in [12]

Figure 4: Few samples of multi-column documents which we considered for this work.



(a) Part of Input Image

(b) Incorrect line segmentation



(c) Automatic error localization

(d) Automatic error correction

Figure 5: Example results: (a) Part of two sample document images (b) Line segmentation output of projection profile based segmentation algorithm. We observe that due to failure in adapting the parameters, segmentation algorithms often fail in few specific regions of documents (c) Output of our automatic error localization module. Green, Blue and Red colours shows correct, under-segmentation and over-segmentation respectively. (d) Output of our error correction module. We observe that "learning by example" based our scheme corrects the errors successfully (best viewed in colour).

and are challenging to segment. We also have line level annotation in form of XML for this dataset, produced using a semi-automatic tool. Few sample images of our dataset are shown in Figure 4.

5.1 Text block segmentation

For text graphics separation and block identification we rely on open source software *iLayout*.

iLayout: page layout Engine.

ilayout is build on top of Leptonica [11]. Some of the tasks which we do using this engine are:

- Multi-column segmentation
- Text and graphics separation.
- Column ordering.

The *iLayout* gives the list of text blocks as output. Once text blocks are identified we use projection profile based segmentation to extract lines. We apply our segmentation post-processor to improve the line segmentation.

Block segmentation performance.

To measure the block segmentation performance of the proposed method we use standard intersection divide by union score with ground truth. Mathematically, let A and B be the set of block bounding boxes representing block segmentation output and ground truth rectangles respectively, then goodness score for each block $a \in A$ is defined as, $S(a) = max\{(a \cap b)/(a \cup b) : \forall b \in B\}$. Thus for a ideally segmented block $S(a) = 1$ and for a false alarm $S(a) = 0$. In other words, value of $S(a)$ close to 1 shows better block segmentation. We observe that around 90% of the blocks have high goodness score *i.e.* $S(a)$ greater than 0.9. Figure 6 shows the qualitative example of text-graphic separations and text block identification. These examples are taken from a large corpus of Indian language document image dataset. As can be seen *iLayout* not only correctly separates the text/graphics but also identifies the text blocks accurately. (Note that many of the existing implementations fail to identify block accurately for this dataset). A failure example at *iLayout* based text-block segmentation level is shown in Figure 7. The failure occurs mainly due to splitting or merging of two consecutive paragraph. However, such failure does not affect the overall OCR accuracy.

5.2 Text line Segmentation

We first run a projection profile based segmentation algorithm on all text blocks and evaluated its performance. we observe that on average around 11% of the segmented lines are not correct. The goal of this work is to automatically localize and correct these errors. Since these errors deteriorates the overall accuracy of an OCR system, correcting these errors can be considered one of the major steps towards improving OCR accuracy.

We, then automatically localize text line segmentation errors. For this we use a training set of around $10K$ text-lines (independent of test pages). We compute the features described in Section 4.2 and train 1 vs rest SVM classifier with RBF kernel on it. (Recall that we formulate the automatic localization of segmentation errors as multi-class classification problem, having five classes namely correct, over-segmentation, under-segmentation, false alarm and missing component). We obtain the text line segmentation error localization accuracy of 85.22%. In other words, around 85.22% of the text-line segmentation errors are localized without using ground truth. Note that identical to [12] when we measure overall error localization accuracyρ_l we also consider percentage of correct lines classified as correct. In other words, Let C be a confusion matrix showing confusion among correct, over-segmentation, under-segmentation, missing component and false alarm. Further, suppose rows $\{0, 1, 2, 3, 4\}$ of C correspond to correct, over-segmentation, under-segmentation, missing component and false alarm respectively. Then we define overall error localization accuracy at line level as follows:

$$\rho_l = \frac{\sum_{i=1}^{4} C_{ii} \times 100}{\sum_{i=1}^{4}\sum_{j=0}^{4} C_{ij}}$$

correct	overseg	underseg	m.c.	f.a.	ρ_l
99.11	**69.65**	**88.00**	**68.22**	**70.12**	**85.22**

Table 2: Percentage of segmentation errors we automatically detect (Proposed Scheme). Note that entry under correct column (99.11%) shows that we tag *correct lines as correct* with a very high accuracy of 99.11%. Here ρ_l denotes the overall error localization performance.

Error localization performance is summarized in Table 2.

Once errors are localized, we correct them using our correction module. We show improvement in accuracy in Table 1. We observe that the proposed segmentation post-processor significantly reduces the segmentation errors present in the segmented lines. These error reduction results around 5% improvement in the overall text-line segmentation.

In these experiments, we have shown improvement in text-line segmentation for a projection profile based segmentation algorithm. However, the method used in designing the segmentation post processor is not specific to any segmentation algorithm. Thus the proposed segmentation post processor can be applied to any text-line segmentation algorithm or the proposed segmentation post-processor is algorithm independent. Further, rather than relaying on heuristic parameters, proposed method works based on a supervised learning framework, where segmentation errors are learnt from the examples. This also makes the proposed post-processor dynamic to variety of pages. Few example results of the proposed method are shown in Figure 5. The proposed method automatically localizes the segmentation errors and corrects them.

6. CONCLUSIONS

A simple but powerful method for correcting text line segmentation errors is proposed. The method uses training examples to localize and correct errors, and thus is dynamic and applicable to a wide variety of documents. We have shown significant improvement in text line segmentation on a large dataset of Indian language scanned document images.

Language	Over Segmentation		Under Segmentation		Missing Component		False Alarm	
	Before	After	Before	After	Before	After	Before	After
Telugu	5.82	**2.12**	0.75	**0.51**	4.9	**2.1**	0.9	**0.4**
Tamil	2.80	**1.52**	4.12	**2.26**	1.6	**0.8**	1.2	**0.78**
Malayalam	0.46	**0.38**	0.5	**0.4**	0.95	**0.42**	0.65	**0.31**
Kannada	3.15	**2.09**	3.61	**2.58**	2.48	**1.10**	1.9	**0.98**

Table 1: Segmentation error reduction (in %) when our segmentation post-processor is applied. Before and after column show errors before and after applying our post processor. It can be seen that the proposed error localization reduces around 5% of the line segmentation errors

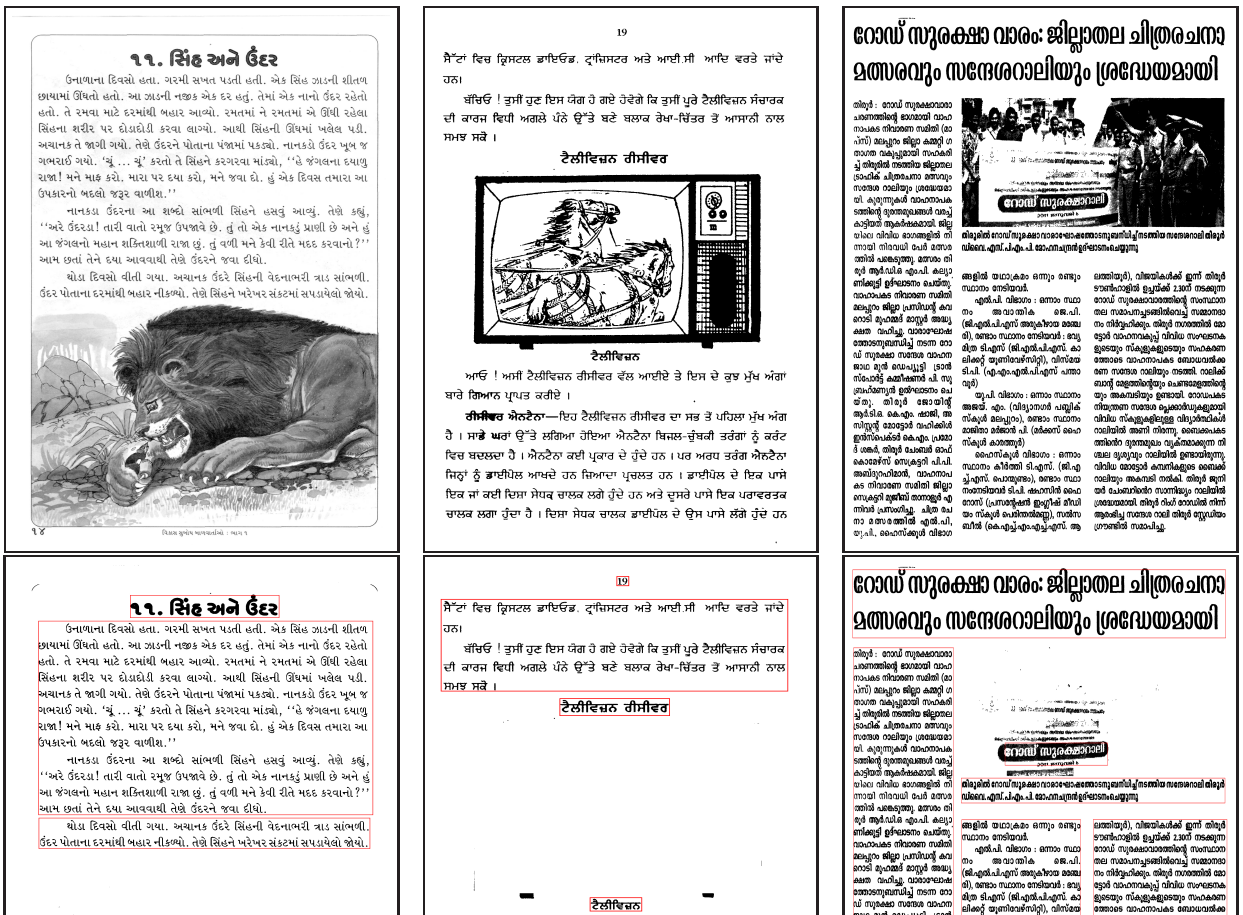


Figure 6: Text-Graphics separation and Text-Block identification by *i-layout*. First and second row shows input and output image respectively. Graphics are removed and text-blocks are identified in red (best viewed in colour).

Acknowledgements..

This work was partly supported by the MCIT, Govt. of India. Anand Mishra is supported by Microsoft Corporation and Microsoft Research India under the MSR India PhD fellowship award.

7. REFERENCES

[1] M. Agrawal and D. S. Doermann. Voronoi++: A dynamic page segmentation approach based on voronoi and docstrum features. In *ICDAR*, 2009.

[2] A. Antonacopoulos, D. Bridson, and B. Gatos. Page segmentation competition. In *ICDAR*, 2005.

[3] A. Antonacopoulos, B. Gatos, and D. Bridson. Page

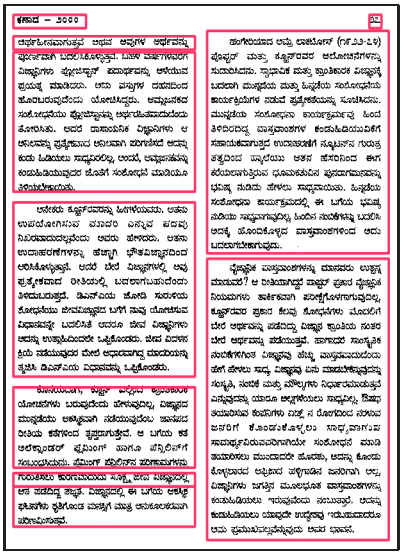


Figure 7: Failure case in Block separation. First paragraph in first column is split into two blocks. Similarly Last paragraph in first column is split into three blocks. Although, this is considered as error for *iLayout* analysis purpose, but fortunately does not affect the OCR accuracy.

segmentation competition. In *ICDAR*, 2007.

[4] A. Antonacopoulos, S. Pletschacher, D. Bridson, and C. Papadopoulos. Icdar 2009 page segmentation competition. In *ICDAR*, 2009.

[5] H. S. Baird, S. E. jones, and S. J. Fortune. Image segmentation by shape-directed covers. In *ICPR*, 1990.

[6] V. Govindaraju and S. Setlur. *Guide to OCR for Indic Scripts*. Springer, 2009.

[7] C. V. Jawahar and A. Kumar. Content-level annotation of large collection of printed document images. In *ICDAR*, 2007.

[8] K. Kise, A. Sato, and M. Iwata. Segmentation of page images using the area voronoi diagram. *CVIU*, 1998.

[9] V. K. Koppula and A. Negi. Fringe map based text line segmentation of printed telugu document images. In *ICDAR*, 2011.

[10] K. S. S. Kumar, S. Kumar, and C. V. Jawahar. On segmentation of documents in complex scripts. In *ICDAR*, 2007.

[11] http://www.leptonica.com/.

[12] D. Mundhra, A. Mishra, and C. V. Jawahar. Automatic localization of page segmentation errors. In *J-MOCR-AND (ICDAR Workshop)*, 2011.

[13] G. Nagy, S. C. Seth, and M. Viswanathan. A prototype document image analysis system for technical journals. *IEEE Computer*, 1992.

[14] L. O'Gorman. The document spectrum for page layout analysis. *IEEE TPAMI*, 1993.

[15] F. Shafait, D. Keysers, and T. M. Breuel. Performance evaluation and benchmarking of six-page segmentation algorithms. *IEEE TPAMI*, 2008.

[16] K. Y. Wong, R. G. Casey, and F. M. Wahl. Document analysis system. *IBM Journal of research and development*, 1982.

[17] H. Zhang, J. E. Fritts, and S. A. Goldman. Image segmentation evaluation: A survey of unsupervised methods. *CVIU*, 2008.