Splitting Merged Characters of Kannada Benchmark Dataset using Simplified Paired-Valleys and L-Cut

H. R. Shiva Kumar, A. Madhavaraj and A. G. Ramakrishnan
Dept. of Electrical Engineering, Indian Institute of Science, Bangalore, India
shivahr@gmail.com, madhavaraja@iisc.ac.in, agr@iisc.ac.in

***Abstract*—We reduce the computational complexity of the paired-valley algorithm for splitting merged characters, from $\Theta(N^2)$ down to $\Theta(N)$, where N is the number of symbols merged. We also propose an effective way (L-cut algorithm) to separate the merged half-consonants (known in Kannada as *ottus*) from the base symbols. We have created a benchmark dataset of 4033 sub-word images in Kannada, each comprising two or more merged characters. We test the recognition accuracy of Tesseract OCR on the created benchmark dataset, before and after applying our technique. The accuracy of Tesseract v3 OCR on the created dataset of 61.6% increases by 20% to a value of 81.7% after the splitting of the characters by our method. The algorithm's scalability to other scripts has been explored by limited experiments on Telugu and Tamil.**

***Index Terms*—Merged characters, printed text, paired valleys, Kannada, ottu, Tamil, Telugu, OCR, VPP, Tesseract, old books, computational complexity.**

I. Introduction

Merged characters that occur in old printed books are unseen patterns for the classifier and reduce the performance of optical character recognition (OCR) systems. They are akin to the occurrence of the out of vocabulary words in automated speech recognition systems. A number of techniques have been proposed to segment and recognize the merged characters, and can be basically classified as recognition-free and integrated segmentation-recognition (ISR) approaches. In the former approach, a set of rules is used to segment the characters before recognition, whereas in the latter approach, the recognition scores for the segmented components returned by the classifier are used to choose one out of many candidate segmentation paths.

A. Literature Survey

Zhu et al. [1] and Liu et al. [2] used ISR method to separate merged handwritten characters in Japanese and Chinese, respectively. Yang et al. [3] used the Vertical projection profile (VPP) of merged Chinese characters to obtain candidate cut locations (CCL) and chose the optimal ones using the recognition score. Messelodi and Modena [4] also split merged Roman characters using VPP. Davessar et al. [5] first vertically cut the merged Gurmukhi characters in the middle and searched for the CL within a window and confirmed it using recognition feedback and the aspect ratios of the separated units. Bayer et al. [6] employed a statistical cut classifier and a search procedure to identify the merge locations in printed text, wherein, the computational complexity increases disproportionately with the number of merged characters in the image. Wang and Jean [7] deployed a neural network and shortest path to segment merged characters. Zhang, Tian and Li [8] use contour analysis to extract the concave points in the merged images of mathematical symbols and use them to postulate cut locations (CL) and employ a recognizer to verify them. Employing the histogram of the merged image to form binary-tree indexed demarcation using forward-backward algorithm, Tang et. al. [9] split the merged symbols.

Most of the recognition-free segmentation approaches proposed in the literature have dealt with merges in handwritten characters only. Congedo et al. [10] put forth the drop fall algorithm to split merged numeric strings in handwritten documents. Chang et al. [11] obtained the convex hull of the merged character and segmented merges in printed text using the features obtained from the concave residual and the shortest path algorithm. Lacerda and Mello [12] chose the optimal cut locations for splitting digit string merges in handwriting, employing the image skeleton and self-organizing maps.

Madhavaraj et. al. [13] reported the maiden effort in splitting of merged Kannada characters. They proposed a recognition based method for segmenting merged characters in printed Kannada documents by pairing top and bottom valleys of the merged-character image to locate the candidate cut locations (CCLs). Further, aspect ratios of the segmented parts, their recognition labels and scores returned by the classifier are used to choose the best segmentation path (SP). We refer to their algorithm as paired-valleys based ISR (PV-ISR).

B. Computational Complexity of PV-ISR method

A major computational complexity of the PV-ISR method is the need for exhaustive search for the optimal segmentation path from all possible paths that can be hypothesized from the candidate cut locations. For example, for the merged image shown in Fig. 1, the PV-ISR algorithm detects 7 CCLs, namely P1 to P7. The possible segmentation units (SU) are {B-P1, B-P2, ..., B-E, P1-P2, P1-P3, ..., P1-E, ..., P7-E} giving a total of 35 possible SUs, excluding input image B-E . Thus, the number of

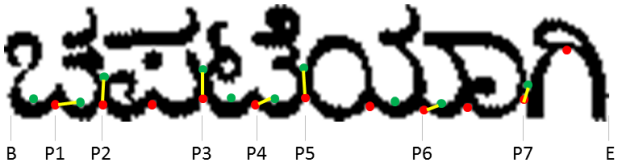


Fig. 1: Example of a Kannada sub-word with merged characters showing the candidate cut locations (CCLs) obtained from the PV-ISR algorithm. Top valley points are shown in green color and bottom ones, in red. Yellow lines show the CCLs, where there are matching pairs of top and bottom valleys in close proximity.

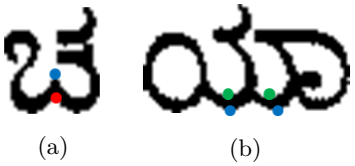


Fig. 2: Filtering out the valley points within a character: (a) Filter out bottom valleys (red color) in the 'ω' shaped portion of the character by looking for matching valleys (blue color) in the negative image. (b) Filter out top valleys (green color) in the ⌣ shaped portions of the characters by looking for matching valleys (blue color) in the negative image at the bottom of the character.

distinct SUs is $(N+1)(N+2)/2-1 = N(N+3)/2$, where N is the number of CCLs, and not $(N2+3)/2$ as wrongly mentioned by Madhavaraj et al. in [13]. For each of those SUs, the PV-ISR algorithm extracts the required features and feeds it to a character recognizer. If the aspect ratio (AR) of any SU lies beyond the expected AR range of the recognized character class, then the recognition score of that SU is changed to a minimum value. Next, for finding the optimal SP, it considers the $(2^7 - 1) = 127$ possible SPs, computes the average likelihood of each of them, and finally selects the one with the maximum likelihood score. Further, the PV-ISR algorithm does not deal with merges of the base characters with the *ottus*, which types of merges also occur frequently in old Kannada texts.

C. Contributions of the paper

We simplify the paired-valleys algorithm for splitting of character merges so that it becomes a recognition-free approach and hence significantly reduces its computational complexity from $\Theta(N^2)$ down to $\Theta(N)$. The simplified algorithm can be applied as a pre-processing step before running any OCR, for the splitting of merged characters, as is demonstrated for the Tesseract OCR, in Sec. IV.

We also present a maiden algorithm for the detection and splitting (L-cut algorithm) of the merger of Kannada ottu symbols with the base characters.

For rigorously evaluating the performance of various algorithms, we have created a benchmarking dataset of 4033 Kannada sub-word images, each containing two or more merged symbols, along with their ground truth text in Unicode. Section IV compares the results of VPP-ISR, PV-ISR and our algorithm on this benchmarking dataset.

II. Simplified Paired Valleys and L-Cut (SPV-LC) Algorithm

A. Simplified Paired Valleys based Splitting

Merges in Kannada printed documents generally happen between the middle portions of successive characters, where they usually have outwardly rounded shape. This results in the formation of valleys just above and below the merged portion. Paired valleys based algorithms for splitting merged characters, such as PV-ISR [13], rely upon

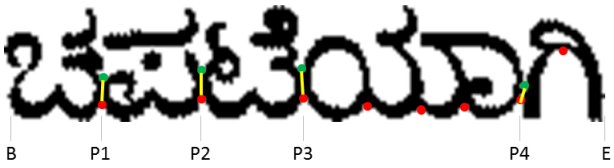


Fig. 3: Cut locations for the Kannada sub-word obtained by our SPV algorithm. Top valleys remaining after filtering are shown in green color and bottom ones, in red. Yellow lines show the cut locations, where there are matching pairs of top and bottom valleys in close proximity.

the detection of such pairs of valley points for determining the candidate cut locations (CCLs). Since valleys are also formed within a character, the CCLs could cut a valid character leading to oversegmentation as shown in Fig. 1 (at P1, P4 or P6). Here, each of the characters ಚ/ca/, ಟೆ/tte/ and ಯಾ/yaa/ are cut into two pieces by the CCLs P1, P4 & P6, respectively. To avoid oversegmentation, recognition based approaches such as PV-ISR, consider all the possible combinations of CCLs, recognize the resulting segmentation units (SUs) and select the optimal sequence of CCLs that gives the maximum recognition score.

We propose a simple but effective approach to avoid oversegmentation by filtering out the valley points formed within a character, thereby solving the problem at the source. Characters in Kannada that have a 'ω' shaped portion at the bottom such as ಚ/ca/, ಜ/ja/, ಟ/tta/, ಡ/dda/, ದ/da/ and ಪ/pa/ have a bottom valley at the mid of the character. Such bottom valleys among the background pixels have a counterpart among the foreground pixels immediately above them, as shown in Fig. 2a. Here, the bottom valley among the background pixels is shown in red color, and its counterpart among the foreground pixels, in blue. Similar pattern exists in characters that have an 'm' shaped portion at the top such as ಣ/nna/ or ಇ/i/, where the top valley among the foreground pixels has a counterpart among the foreground pixels immediately below it. Hence, to avoid oversegmentation, we filter out valley points that have an immediate counterpart among foreground pixels as shown in Fig. 2a.

Kannada characters that have ⌣ shaped portions at the bottom such as ಚ/ca/, ಟ/tta/ and ಯ/ya/ have top valleys at the mid of the ⌣ shape. Such top valleys among the background pixels also have their counterpart among the foreground pixels immediately below them at the bottom of the character as shown in Fig. 2b. Here, the top valleys among background pixels are shown in green color, and their counterparts among the foreground pixels, in blue. Again, we prevent oversegmentation by filtering out such top valleys possessing immediate counterparts among the foreground pixels at the bottom of the character, as shown in Fig. 2b.

Figure 3 shows the valleys detected by our algorithm on the same image shown in Fig. 1. Our algorithm has filtered out the valley points formed due to ω and ⌣ shaped portions, thereby avoiding oversegmentation.

B. Detecting and Splitting Base-Ottu Merges

Another common merge in Kannada characters occurs due to the touching of ottu symbols with the base symbols as shown in Fig. 4. Ottu symbols are additional graphemes used for representing consonant conjuncts and they appear below the *baseline* of the line [14]. Ottus can get merged with the previous consonant or the next one, leading to right-side or left-side merges, as shown in the first and second rows of Fig. 5. Some ottu symbols can appear right below the base consonant as shown in the third row of Fig. 5. We refer to this as the center merge.

An intuitive technique for splitting such merges is to cut the base-ottu merger just below the baseline and additionaly along the left or right side of the base symbol, as shown in the second column of Fig. 5. We refer to this technique as L-cut (LC) due to the resemblance of the cut lines with the 'L' shape or its mirror image.

An important step before we can perform L-cut is the detection of baseline, for which we can leverage horizontal projection profile (HPP), as shown in Fig. 7. We smooth the HPP with a Gaussian function to remove jitter, and then take its first derivative. The location of the minimum of HPP derivative in the lower half of the image is taken as the position of the baseline. The third column of Fig. 5 shows the split of the base from the ottu symbols using L-cut.

To handle the scenarios, where there are merges among base characters as well as base-ottu merges, as shown in the first column of Fig. 6, we first invoke SPV based splitting on the image region above baseline. This splits the merges among the base characters, as shown in the second and third columns of Fig. 6. Then, on each connected component in the SPV split image, we detect if it corresponds to a base-ottu merger by looking at the extent of region below the baseline. If a base-ottu merger is detected, we invoke L-cut as shown in the fourth column of Fig. 6. The fifth column shows the components, after the splitting by the combined SPV-LC algorithm.

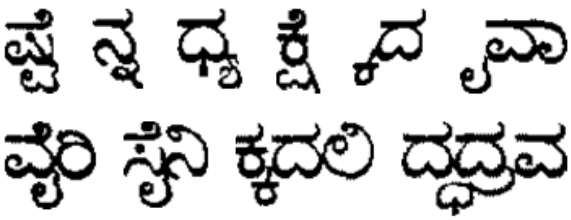


Fig. 4: Some example images, showing typical merges of the base characters (in the middle zone) with the ottu symbols (half consonants occurring below the baseline).

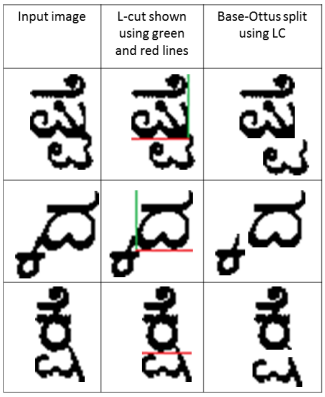


Fig. 5: Some examples of base-ottu merges, separated (split) correctly using the proposed L-cut (LC) algorithm.

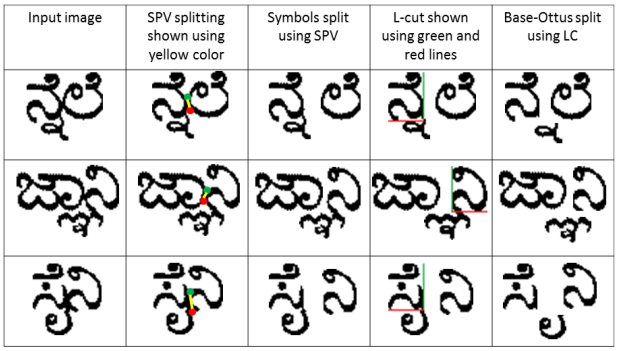


Fig. 6: Illustration of the SPV-LC algorithm, with 3 examples. First, the merged base characters are split by invoking the SPV algorithm on the image region above baseline. Then, the base-ottu merges are split using L-cut.

III. Kannada Dataset for Benchmarking

For rigorously evaluating the performance of the proposed algorithm in splitting character merges, we have created a benchmarking dataset of 4033 sub-word level images, each containing two or more merged symbols. Figure 8 illustrates some of the images from this benchmarking dataset, showing merges across base components, as well as across base and ottu symbols. The ground truth text for each of the test images is provided in a separate Unicode text file as shown in Table I.

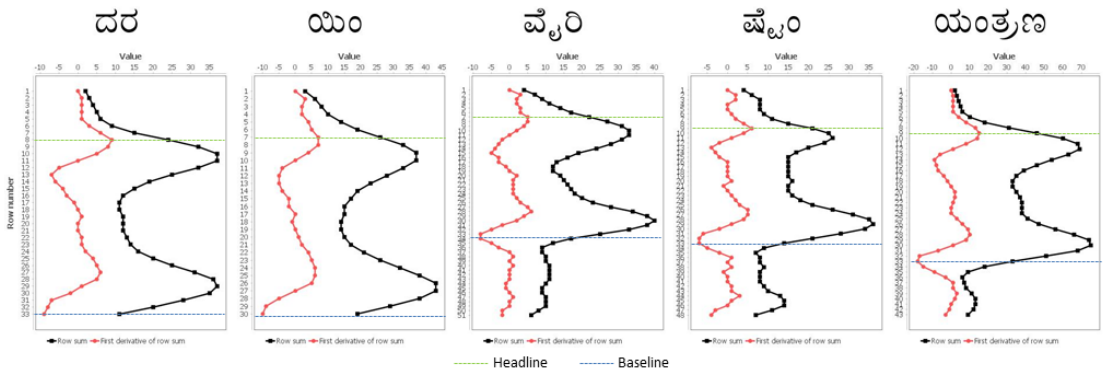


Fig. 7: Detecting headline and baseline using Gaussian smoothed horizontal projection profile and its derivative.

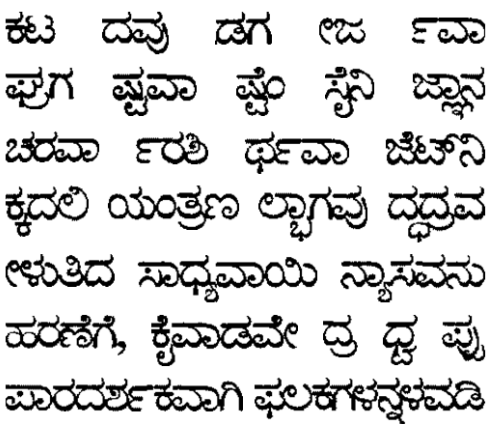
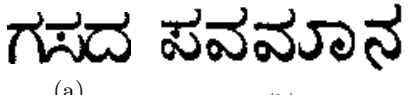


Fig. 8: Samples from the Kannada merged symbol dataset of 4033 sub-word images, each containing two or more merged symbols. Test images contain merges between base components, as well as between base and ottu components.

TABLE I: The ground truth text for each test image in the Kannada merged symbol dataset has been provided in a separate Unicode text file in the following format.

Image Name	Unicode Text
C0032.tif	ಕಟ
C0037.tif	ದವು
C0033.tif	ಡಗ
C0228.tif	೨ ೀಜ
C0441.tif	೯ವಾ
C0142.tif	ಪ್ರಗ
C0117.tif	ಷ್ಟವಾ

Image Name	Unicode Text
C0223.tif	ಷ್ಟೆಂ
C0206.tif	ಸೈನಿ
C0479.tif	ಜ್ಞಾನ
C0008.tif	ಚರವಾ
C0125.tif	೯ರಶಿ
C0199.tif	ರ್ಧವಾ
C0053.tif	ಜೆಟ್ಸಿ



(a) (b)

Fig. 9: (a) A sample Kannada sub-word, with merged symbols. (b) A letter press printed Kannada word.

Performance of the proposed algorithm is evaluated as the Levenshtein distance between the recognized and the ground truth texts, across the entire benchmarking dataset. Let N, S, I and D denote the number of Unicodes in ground-truth, substitutions, insertions and deletions, respectively. The Unicode level recognition accuracy is determined as:

$$Accuracy = (N - S - I - D)/N \quad (1)$$

This approach counts errors from both segmentation (merged symbols that are not split or those that are split incorrectly) and recognition phases. A better approach to measure errors only from the segmentation phase would require pixel-level ground truth of symbols, where all the pixels belonging to each symbol would be given an unique number in ground truth. For example, for sub-word ಗಸದ/gasada/ shown in Fig. 9a, all the pixels of the symbols ಗ/ga/, ಸ/sa/ and ದ/da/ would have values of 1, 2 and 3, respectively. In addition, for Indic scripts such as Kannada, such an approach should also standardize the list of OCR symbols for that script. For example, for the letter press printed Kannada word ಪವಮಾನ/pavamaana/, shown in Fig. 9b, the list of symbols could be {ಪ/pa/, ವ/va/, ಮಾ/maa/, ನ/na/} or {ಪ/pa/, ವ/va/, ವ/va/, ಾ/vowel_sign_aa/, ನ/na/}. However, the Unicode representation of that word is unique, consisting of Unicodes {ಪ/pa/, ವ/va/, ಮ/ma/, ಾ/vowel_sign_aa/ and ನ/na/}. Currently, the created benchmarking dataset has ground truth only at the Unicode/text level.

IV. Results and Discussion

We compare the performance of our SPV-LC algorithm against those of the vertical projection profile (VPP-ISR) and paired-valleys (PV-ISR) based ISR algorithms on the new benchmarking dataset. The VPP-ISR and PV-ISR algorithms, as described in [13], are run on all the 4,033 images in the new benchmarking dataset, and the character recognition accuracy is computed as per eqn. (1). Table II shows the recognition accuracies of VPP-ISR and PV-ISR algorithms. For measuring the performance of our SPV-LC algorithm, we recognize the segmented

TABLE II: Comparison of the Unicode recognition accuracies (in %) of various splitting algorithms - VPP-ISR, PV-ISR and the proposed, on Kannada benchmarking dataset of 4,033 sub-word images, each containing 2 or more merged symbols. N, M: # of Unicodes in ground-truth and OCR output, respectively. N = 15,626. S, I and D: # of substitutions, insertions and deletions.

Algorithm	Accuracy	M	S	I	D
VPP-ISR	63.4	14,419	3,514	496	1,703
PV-ISR	81.0	14,160	1,365	70	1,536
SPV-LC	87.2	15,142	1,017	251	735

characters using the same set of features and classifier as in [13]; the combination of correlation and discrete wavelet transform is used as features and, support vector machine (SVM) with linear kernel is used as classifier. The last row of Table II shows the recognition accuracy of our SPV-LC algorithm. The VPP-ISR, PV-ISR and the SPV-LC algorithms achieve character recognition accuracies of 63.4%, 81% and 87.2%, respectively. The results show that the proposed algorithm achieves the best segmentation of the merged characters. Our results cannot be compared with those of [13], since (i) the database used there consists of complete words, whereas in our case, we use fully merged sub-words and (ii) the database used in [13] is not public.

The impact of L-cut is measured by running the SPV based splitting without using L-cut and measuring the difference in character recognition accuracy on the benchmarking dataset. Table III shows the recognition accuracy without using L-cut and the difference. The L-cut improves character recognition accuracy by 4.9%.

Since Tesseract OCR supports Kannada, we ran Tesseract v3 and v4 on each of the 4,033 sub-word images in the benchmarking dataset, and computed the character recognition accuracy as per (1). Table IV shows that Tesseract v3 achieves an accuracy of 61.6% on the test dataset, showing that this version of Tesseract has some in-built functionality for splitting character merges. However, Tesseract v4, which is based on LSTM (and hence, is good at capturing word-level language model), achieves a far lower accuracy of 29.9% on the sub-word level dataset, and hence we have skipped reporting of its results. Since our algorithm is recognition-free, it can be leveraged as an useful pre-processing step to split character merges before running the Tesseract OCR. Table IV also shows the performance of Tesseract v3 after preprocessing the input images using our SPV-LC algorithm. Accuracy of Tesseract v3 jumps from 61.6% to 81.7%, showing the utility of the proposed algorithm as a pre-processing step.

Figures 10a, 10b and 11 show examples of successful split of merged symbols using our SPV-LC technique. Figures 12a and 12b show examples, where our algorithm is either unable to split the merged symbols, or splits them

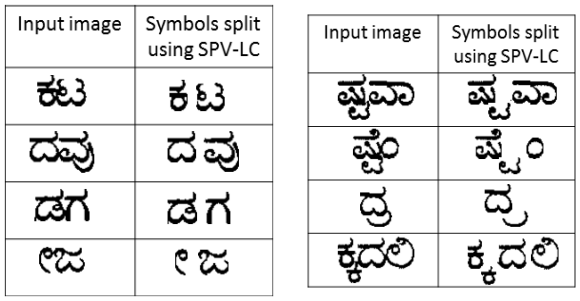
TABLE III: Impact of L-cut: recognition accuracy (in %) without using L-cut and the improvement, due to L-cut. All the notations are the same as in Table II. N = 15,626.

Algorithm	Accuracy	M	S	I	D
SPV without LC	82.3	14,509	1,266	194	1,311
LC improvement	+4.9	+633	-249	+57	-576

TABLE IV: Recognition accuracies (in %) of Tesseract OCR (v3.04.00) on the benchmark dataset, before and after splitting the input images using SPV-LC algorithm. All the notations are the same as in Table II. N=15,626.

	Accuracy	M	S	I	D
Tesseract v3	61.6	12,991	2,543	412	3,047
Tesseract v3 post SPV-LC	81.7	15,734	1,765	602	494

incorrectly. To see if the SPV-LC algorithm works for other Indic scripts, we tested it on a few merged symbols of Tamil and Telugu. Figures 13a and 13b show the outputs for these samples, which prove its effectiveness.



(a) (b)

Fig. 10: Some samples of merged symbols successfully split using SPV-LC technique. (a) Only base character merges. (b) Both base-base and base-ottu merges.

V. Conclusion

The SPV algorithm is successful in splitting merged symbols in a vast majority of cases, while also reducing the computational complexity to $\Theta(N)$ from $\Theta(N^2)$ in ISR approaches, where N is the number of symbols merged. The maiden L-cut algorithm is largely successful in detecting and splitting the merger of Kannada ottu symbols with the base characters. On the Kannada benchmark dataset, the Unicode recognition rate of Tesseract OCR increases from 61.6% to 81.7% after the splitting of the characters by our method. The algorithm's scalability to other scripts has also been explored by limited experiments on Telugu and Tamil. Thus, it holds promise as a useful module during character segmentation in existing OCRs. The standard, annotated database that has been created by us is now available for researchers [17].

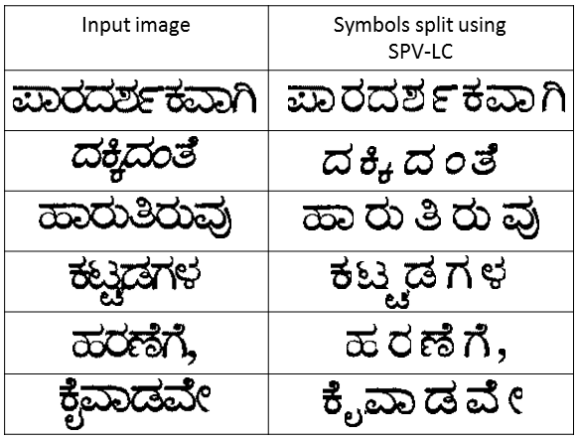
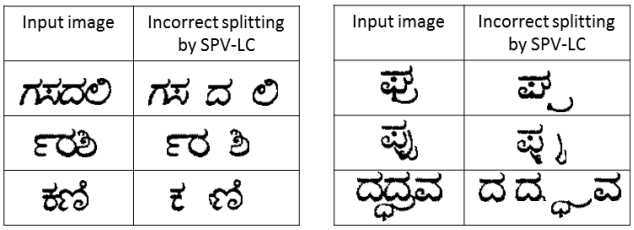
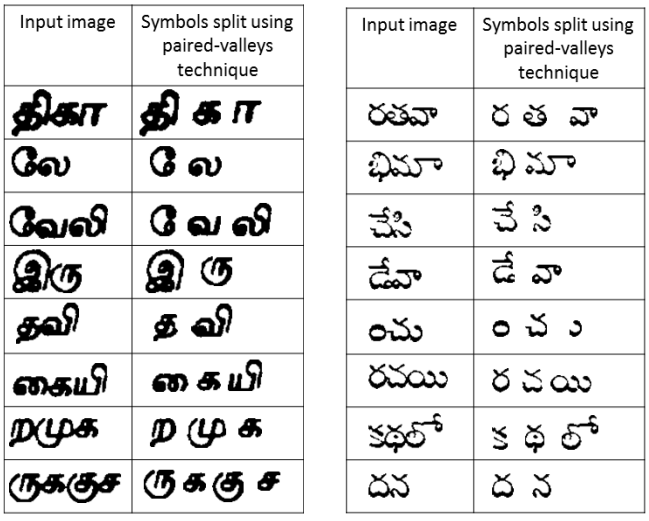


Fig. 11: Some more examples of merged symbols successfully split by the SPV-LC technique.



(a) (b)

Fig. 12: Inadequacy of SPV-LC algorithm. (a) Example images, where SPV technique failed or split the base symbols incorrectly. (b) Examples, where L-cut failed to correctly segment base-ottu merges.



(a) (b)

Fig. 13: Testing of SPV-LC algorithm on other scripts. (a) Sample cases of successful splitting of Tamil merged symbols. (b) Splitting of merges between Telugu symbols.

References

[1] Zhu, B., Zhou, X. D., Liu, C. L., and Nagakawa, M., "A robust model for on-line handwritten Japanese text recognition," Int. Jounal Document Analysis and Recog. 13(2), pp.121-–131, 2010.

[2] Liu, C. L., Jaeger, S. and Nakagawa, M., "Online recognition of Chinese characters: the state-of-the-art," IEEE Trans. Pattern Analysis and Machine Intelligence, 24(2), pp.198-–213, 2004.

[3] Wuyi Yang, Shuwu Zhang, Haibo Zheng and Zhi Zeng, "A recognition-based method for segmentation of Chinese character in images and videos," Proc. Int. Conf. Audio, Language and Image Processing, ICALIP July 2008, pp.723-728.

[4] S. Messelodi and C.M. Modena, "Context driven text segmentation and recognition", Pattern Recognition Letters, Vol. 17(1), pp. 47-56, Jan 1996.

[5] Davessar, N.M., Madan, S. and Hardeep Singh, "A hybrid approach to character segmentation of Gurmukhi script characters," Proc. 32nd Applied Imagery Pattern Recognition Workshop, Oct 2003, pp.169–173.

[6] T. Bayer, U. Krebel and M. Hammelsbeck, "Segmenting merged characters", Proc. XI Int. Conf. Pattern Recognition, Vol. II. Conf. B: Pattern Recognition Methodology and Systems, 1992.

[7] Jin Wang and Jack Jean, "Segmentation of merged characters by neural networks and shortest path", Pattern recognition, Elsevier, Volume 27, Issue 5, May 1994, pp. 649-–658.

[8] Dong-Yu Zhang, Xue-dong Tian and Xin-fu Li, "An improved method for segmentation of touching symbols in printed mathematical expressions," Proc. 2nd Int. Conf. Adv. Computer Control (ICACC) March 2010, vol.2, pp. 251-25.

[9] Tang Y, Li X, Zhang Y, Li M, Xu M, "Segmentation of touching characters via tree-indexed demarcation using forward and backward searches," Adv. in Mech. Engineering. Oct 2017; 9(10).

[10] Congedo, G., Dimauro, G., Impedovo, S. and Pirlo, G., "Segmentation of numeric strings," Proc. Third Int. Conf. Document Analysis and Recognition, Aug 1995, vol.2, pp.1038-1041.

[11] Chang, T.C. and Chen, S.Y, "Character segmentation using convex-hull techniques". Int. J. Pattern Recognition and Artificial Intelligence, vol. 13, no. 6, pp. 833-858, 1999.

[12] Lacerda, E.B. and Mello, C. A B, "Segmentation of touching handwritten digits using self-organizing maps," Proc. 23rd IEEE Int. Conf on Tools with Artificial Intelligence, 2011, pp.134-137.

[13] Madhavaraj A., A. G. Ramakrishnan, Shiva Kumar H. R. and Nagaraj B., "Improved recognition of aged Kannada documents by effective segmentation of merged characters," Proc. Tenth Int. Conf. Signal Processing and Communications, 2014.

[14] Rituraj Kunwar, Shashi Kiran and A. G. Ramakrishnan. "Online handwritten Kannada word recognizer with unrestricted vocabulary," Proc. Int. Conf. on Frontiers in Handwriting Recognition (ICFHR), IEEE, 2010, pp. 611–616.

[15] Vijay Kumar B and A. G. Ramakrishnan. "Machine recognition of printed Kannada text." International Workshop on Document Analysis Systems. Springer, Berlin, Heidelberg, 2002.

[16] Vijay Kumar B and A. G. Ramakrishnan. "Radial basis function and subspace approach for printed Kannada text recognition." IEEE International Conference on Acoustics, Speech, and Signal Processing 2004 May 17 (pp. V-321).

[17] MILE, IISc "Kannada benchmarking dataset of merged symbols" https://github.com/MILE-IISc/MergedSymbolsKannada 2018.

[18] H. R. Shiva Kumar and A. G. Ramakrishnan, "Gamma Enhanced Binarization - An adaptive nonlinear enhancement of degraded word images for improved recognition of split characters," Proc. NCC 2019.