# ANALOGIC PREPROCESSING AND SEGMENTATION ALGORITHMS FOR OFF-LINE HANDWRITING RECOGNITION

GERGELY TÍMÁR, KRISTÓF KARACS AND CSABA REKECZKY

*Analogical & Neural Computing Laboratory,*
*Computer and Automation Research Institute,*
*Budapest, Kende u. 13-17, H-1111, Hungary*
*phone: 36 1 2095357, e-mail: timar@sztaki.hu*

This report describes analogic algorithms used in the preprocessing and segmentation phase of off-line handwriting recognition tasks. The discussed handwriting recognition approach is segmentation based i.e. it attempts to segment the words into their constituent letters. In order to improve their speed the utilized CNN algorithms use dynamic, wave front propagation-based methods instead of relying on morphologic operators embedded into iterative algorithms. The system first locates the handwritten lines in the page image then corrects their skew as necessary. Afterwards it searches for the words within the lines and corrects the skew at the word level as well. A novel trigger wave-based word segmentation algorithm is presented which operates on the skeletons of words. Sample results of experiments conducted on a database of 25 handwritten pages are presented.

## 1 Introduction

This paper is concerned with solving the preprocessing and letter segmentation tasks of an offline handwriting recognition system without the use of a grammatical model or input from the character recognizer. In this context, offline means that the writing is processed after it has been created, for example by scanning in a handwritten page. For a thorough overview of handwriting recognition approaches, the reader is referred to [1,4]. The basic structure of an offline handwriting recognition system is shown in Figure 1. In almost every procedure of the preprocessing algorithms tens or even hundreds of image processing operations are performed on the input images. This is one of the most computationally intensive parts of offline handwriting recognition and motivated the use of the fastest possible hardware architectures (i.e. CNNs [9,10]) during the design of the algorithms.


Figure 1. Block diagram of an offline handwriting recognition system

## 2 The Developed Algorithms

We have been careful to utilize only linear and nearest-neighbor templates that can be efficiently run on existing hardware implementations. A comprehensive flowchart of all algorithmic steps is shown on Figure 2, where the steps implemented on CNNs are shown in italic, and the ones in normal type use traditional digital methods. The utilized CNN templates may be found in the Appendix. The resolution requirements of the algorithms vary, but all single word algorithms require less than 128x128 pixels.

Our line localization algorithm is similar to the ones used in OCR systems [2] where lines are localized by computing the horizontal histograms for the entire image at a couple of relevant skew angles; then the angle and position where the histograms have local minima are chosen as the location between the lines. We refined this algorithm in a number of ways. Based on [12] we compute an approximation of the pseudo convex hull of each word using the **HOLLOW** template [11]. This template computes the convex hull of the objects in the image if it is run sufficiently long (i.e. till there are no changes on the image). If it is stopped "some time" earlier, then the hull will only be pseudo convex. The running time of the template (approximately $37\tau$) was found by experimentation and appears to be fairly consistent across the different images.

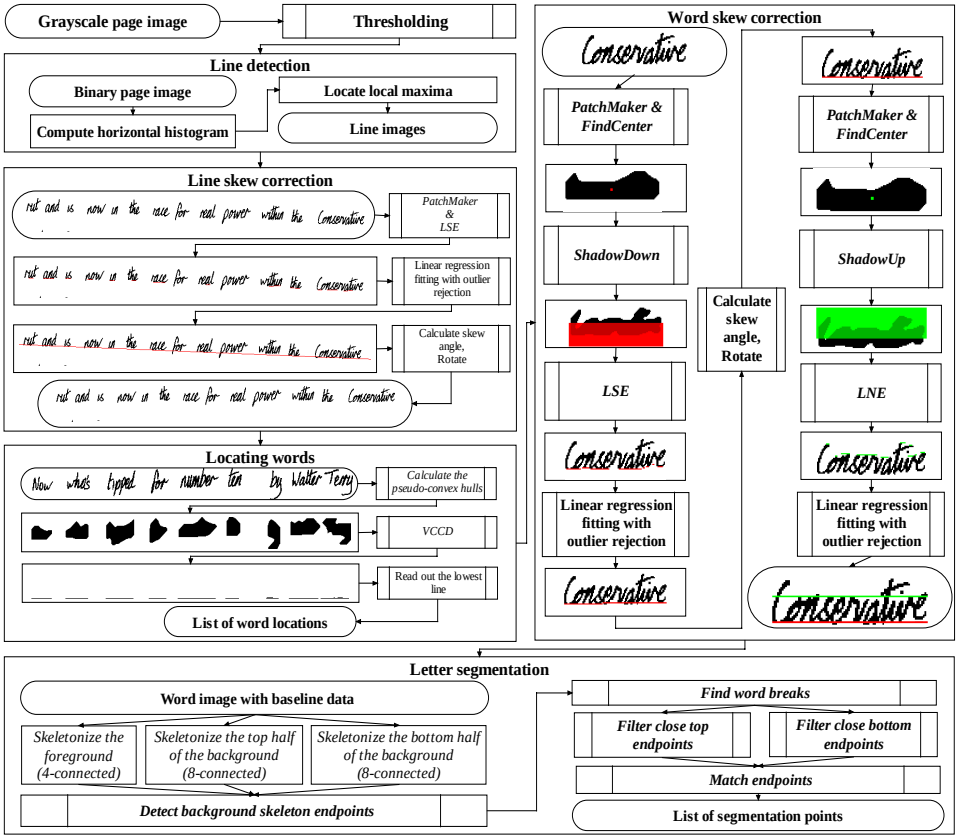


Figure 2. Flowchart of the described preprocessing and segmentation algorithms (CNN algorithms are shown in italic)

The horizontal histogram computed on the pseudo convex hulls is smoothed further via sliding-window averaging with a window size ($p_1$) of 10 (it is effectively low pass filtered). The next step of the algorithm is to find the local maxima of the histogram since these correspond to the location of the lines. Sometimes more than one closely spaced local maxima correspond to the same line (usually because the skew of a line is substantial). To correct this we introduced a second parameter ($p_2$) that specifies a threshold within which we associate all maxima with one line. This parameter is held now at 80% of the largest local maximum. Finally, we drop those maxima that are smaller than a given per-

centage of the average local maxima ($p_3$=25%). By adjusting the parameters $p_1$, $p_2$ and $p_3$ the sensitivity of the algorithm can be varied widely.

After we have localized the lines in a given image, it is possible to correct their skew to somewhat normalize the word images. The skew correction algorithm first finds the lowest points of the pseudo convex hulls using the **LOCAL SOUTHERN ELEMENT (LSE)** detector template from [11]. It uses digital linear regression with outlier rejection to fit a line to remaining black pixels, to calculate the angle of that line and to rotate the original image with that angle.

## 3 Segmentation Of Lines Into Words

To aid further grammatical processing, context analysis and later reconstruction of each run of the algorithm, the lines are stored separately and in sequence as they appear on a page. Relevant information is also stored with each line, such as its original image, the de-skewed image, data about the constituent words etc.

The next step is to locate the words on each line (to exploit context information). This could be easily achieved with the calculation of vertical histograms and identifying the local minima in those histograms, but we looked for an analogic approach that provides almost the same results in this particular application. The conceived algorithm is as follows: for every line, we compute the pseudo convex hull, and then apply the **VCCD** template [11]. This template detects the vertically connected components in an image by shifting them downwards until they disappear. The result of the template for these types of images is that horizontal lines appear in the last row of the image corresponding to the largest horizontal extent of the word images (since there is usually only one vertically connected component, i.e. a word). This makes it possible to extract the word images from the line very easily. The location where the pseudo convex hulls of the words overlap can be identified because there will be two parallel horizontal lines where the overlap occurs.

## 4 Segmentation Of Words Into Letters

In segmentation-based systems, where the basic units of recognition are letters, it is crucial to segment the words into letters as accurately as possible. Unfortunately, this is almost impossible to do correctly as illustrated by the Sayre paradox [7]: “To recognize a letter, one must know where it starts and where it ends, to isolate a letter, one must recognize it first”. This problem can be circumvented by over-segmentation while under-segmentation must be avoided. Unfortunately, many times the word image by itself does not contain enough information to correctly segment it into letters, as shown on Figure 3.

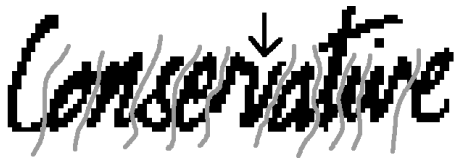

Figure 3. An example for a word that cannot be correctly segmented based on the word image alone (the word is “Conservative”)

Calculation of the word baselines allows the feature extractor to use vertical location of the letters as a feature, so we calculate the word baselines with an algorithm similar to the one in [2]. The pseudo convex hulls are created as before, but a mask needs to be generated to exclude parts of the word where only erroneous elements would be located (the upper half, when computing the lower baseline, and the lower half for the upper baseline). This is not needed when calculating the baseline for the whole line because there are so many local southern elements on a line (since it is much longer), that the regression fitting will still be accurate. We generated this mask by letting the **HOLLOW** template run longer (for approximately $58\tau$), running the **FINDCENTER** CNN algorithm [11] on the result, and then generating a horizontal shadow from the row of the center point using the **SHADOWUP** or **SHADOWDOWN** templates. The result is a mask that can be used to specify where the following templates should be applied. Afterwards, not only the local southern, but also the local northern elements (LNE-s) have to be calculated (with the template **LNE**) because the upper baselines are needed too.

The letter segmentation algorithm developed by us is based on the same ideas as the segmentation algorithm described for touching numerals in [6]. The basic idea is that we can gain meaningful information about a word's structure by skeletonising it and its background, and locating the junction- and endpoints of the skeleton. By using these special points, we are (mostly) able to construct the segmentation boundaries.

A necessary condition for a correct letter segmentation boundary is that it should start above the foreground skeleton (i.e. the word) and end under it, and it may only cross the foreground skeleton once. If we skeletonize the whole background at the same time, identifying which endpoints are above and below the word skeleton becomes an extra and not so trivial task. If the foreground skeleton image is 4-connected, then the skeleton can be used as a barrier between the upper and lower parts of the background skeleton. This can be exploited by flood-filling the background from the upper and lower baselines at the same time. This ensures that if there are breaks in the word then the flood won't "overflow". These floods can then be separately skeletonised, and the characteristic points of the skeleton identified. Flood filling can be very efficiently accomplished with trigger waves on CNNs. A thorough description of trigger waves with CNNs, their properties and some applications can be found in [8]. The template used for filling the background is the **BPROP** template.

The skeletonization algorithm described in [11] is suitable for general purposes but in this particular application, the structure of the resulting skeleton is crucial, so we experimented with different ordering of the skeletonization templates. This has a huge impact on the final skeleton so we used the template order **SKELE5-7-1-3, SKELE6-8-2-4,** and **SKELE1-2-3-4-5-6-7-8.**

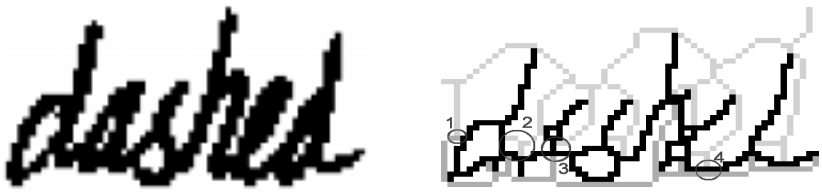


Figure 4. Segmentation heuristics using skeleton endpoints (the word is: "dashed")

The foreground skeleton must be 4-connected for the above algorithm to work and skeletonising with the **SKLHV1-2-3-4-5-6-7-8** templates can ensure this. However, if the original word image was itself not 4-connected, then the resulting skeleton won't be 4-

connected either, so the original image has to be preprocessed to ensure 4-connectedness. This can be done using the **CONN4SE**, **CONN4SW**, **CONN4NE** and **CONN4NW** templates that fill a given pixel according to local rules.

After studying the skeletonisation images of handwritten words, we have found only two requirements that have to be filled by a correct segmentation boundary: no endpoint can be part of two boundaries at the same time and a segmentation boundary must start at a top endpoint and end at a bottom endpoint. Unfortunately these are not enough to filter and match the already identified skeleton endpoints so some other heuristics are needed, which are illustrated on Figure 4.

At location 1, the top and a bottom endpoints are located right next to each other so it is surely a segment boundary, since this can only happen where the top and bottom floods have met because of a word break (here it signifies the start of the first letter). Location 2 shows an instance where the skeleton needs postprocessing to unite these short branches that add only clutter to the problem. A novel approach to solve this is presented in the next section. Location 3 shows the average case where the top and bottom endpoints are located close to (within a specified distance *d*), but not right next to each other. There is a boundary point where the shortest path connecting these points intersects the word skeleton. Finally location 4 shows a special case that is quite common, when there is only a top endpoint with a horizontal (or near horizontal) line segment on the lower background skeleton. This is usually a segmentation point, but has to be found separately from the other cases.

## 5 Postprocessing The Endpoints Of Skeletons

The objective of the algorithm is to replace the endpoints of two short branches of a skeleton with one endpoint located approximately halfway between the original points and on the same side of the word skeleton. This problem can be efficiently solved using trigger waves with isotropic templates, which can be used to measure distance on and image. To accomplish this, we detect whether the two wave fronts generated by **CPATCH** met while the waves propagated for a specified time (say *t*), by running the inverse template (**CWPATCH**) for the same amount of time (*t*). In all places where the patches stayed convex only the original starting pixels will remain, but where the wave fronts did merge, multi-pixel patches will be visible. This is illustrated on Figure 5. We can give an upper estimate of the distance based on the running time of the templates. The "skeleton short branch merger" algorithm proceeds as described, but uses the center points of the multi-pixel patches as the merged point between too close ones.

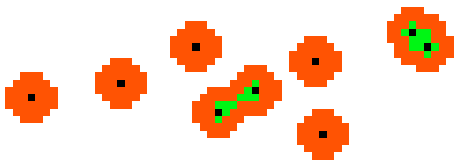

Figure 5. Distance approximation with trigger waves. The black points are the starting points; the grey circles show the maximal extent of the wave fronts (after CPATCH) and the light gray patches the result after applying the CWPATCH template

## 6 Steps Of The Letter Segmentation Algorithm

1. Find the skeleton endpoints.
2. Merge the endpoints on short branches, separately for the top half of the background skeleton and for the bottom half.
3. Find the word breaks, i.e. those locations, where the top endpoints and bottom endpoints are right next to each other (location 1 in Fig. 2). Store these as segmentation boundaries and remove them from further processing.
4. Find those skeleton endpoint pairs where one is on the top skeleton, the other on the bottom skeleton and they are closer to each other than some predefined distance *d* (location 3). This can be accomplished by the endpoint post-processing algorithm, and running **FINDCENTER** on the intersection of the foreground skeleton and the patches remaining. Add the center points to the segmentation boundaries and remove them from further processing.
5. From the remaining top endpoints generate vertical lines that are at most *n* pixels long using the **DPROP** template. If these reach the bottom skeleton and the lines intersect the foreground skeleton, then add the lowest intersection points to the segment boundaries.

## 7 Discussion Of The Results

The described algorithms were tested on a database of 25 handwritten pages collected by Andrew Senior [3,5]. The database contains 7000 words from the LOB corpus (Lancaster-Oslo / Bergen) that were written by a single writer. Since the segmentation algorithms do not distinguish punctuation marks from real words, and the segmentation files do not contain punctuation information either, we have erased the punctuation marks from the page images.

We have run these line, word and letter segmentation algorithms on 5 pages of the 25-page database. The line segmentation algorithm found every line on every page correctly. Statistics for the word segmentation algorithm are:

| | |
|---|---|
| Total number of lines processed: | 85 |
| Correctly segmented lines: | 74 (87.06 %) |
| Incorrectly segmented lines: | 11 (12.41 %) |
| Lines w. more words than expected: | 8 (9.41 %) |
| Lines w. fewer words than expected: | 3 (3.53 %) |

These results are quite good, since 87% of the lines were segmented correctly into words, and out of the erroneous 11 there were only 3 lines with less words than expected. This is important, because merging words afterwards is easier than splitting up words. This means, that 96.5% of the lines were either correctly segmented, or can be easily corrected.

Assessing the results of the letter segmentation algorithm is not as straight forward as the other two's. The problem is that there are many equivalent segmentation boundaries, which cannot be identified easily by a computer program; one has to inspect the word images manually, one by one, which requires a great deal of time. We decided to evaluate the algorithm differently. Since the goal was over-segmentation we can assess the algorithm by comparing the number of segmentation boundaries with the number of letters

specified in the segmentation file, which will give us a rough estimate of the effectiveness of the algorithm. These statistics are:

| | |
|---|---|
| Total number of words found: | 327 |
| Total correctly (over)segmented words: | 247 (75.54 %) |
| Total incorrectly (under)segmented words: | 80 (24.46 %) |

We also inspected the segmentation of random words visually to further grade the algorithm. Results for a few words can be seen in Fig. 4. Note, that there are a couple of problem sites (encircled), some of which are difficult to resolve even for a human reader. It is also evident why automatic evaluation of the segment boundaries is next to impossible.

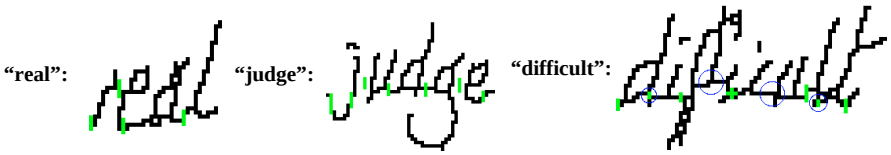


Figure 6. Word recognition results for a few sample words. The segment boundaries have been manually enlarged to be clearly visible. The circles indicate problem spots

## 8 Conclusions

We have demonstrated that analogic algorithms can be utilized effectively in the preprocessing and segmentation problems of off-line handwriting recognition. By avoiding iterative methods and using propagating wave-based approaches where possible, the inherent parallel processing capabilities of CNN arrays can be greatly exploited. It is also evident, that incorporating feedback from the recognition engine could enhance the performance of these algorithms. For example this could help in the letter segmentation by validating some segments that could be recognized with a high confidence level and the remaining segments could be determined with this information in mind. Plans are also underway to integrate a recognition engine into the system.

## 9 Acknowledgements

The authors would like to thank the helpful ideas and support of Dr. Gábor Horváth and prof. Tamás Roska during the development of these algorithms.

## References

1. T. Steinherz, E. Rivlin and N. Intrator: Offline cursive script recognition – a survey, *International Journal On Document Analysis And Recognition*, Vol. 2, pp.90-110, 1999.
2. S. N. Shrihari et al.: „Analysis of Textual Images Using The Hough Transform", *Machine Vision and Applications*, Vol. 2, pp.141-153, 1989
3. A. Senior and A.J Robinson: „An Off-line Cursive Handwriting Recognition System", *IEEE Transactions On Pattern Matching And Machine Intelligence*, Vol. 20, pp. 309-321, 1998.

4. R. Plamondon and S. Shrihari: „ On-Line and Off-Line Handwriting Recognition: A Comprehensive Survey", *IEEE Transactions On Pattern Matching And Machine Intelligence*, Vol. 22, pp. 63-84, 2000.
5. A. Senior LOB Database: ftp://svr-ftp.eng.cam.ac.uk/pub/reports/Senior_tr105.ps.Z
6. Y. Chen and J. Wang: „Segmentation of Single- or Multiple-Touching Handwritten Numeral String Using Background and Foreground analysis", *IEEE Transactions On Pattern Matching And Machine Intelligence*, Vol. 22, pp.1304-1317, 2000.
7. Sayre: „Machine Recognition of Handwritten Words: A Project Report", *Pattern Recognition*, Vol. 5, No. 3, pp. 213-228, 1973.
8. Cs. Rekeczky, L. Chua : „Computing with Front Propagation: Active Contour and Skeleton Models in Continuous-Time CNN", *Journal of VLSI Signal Processing*, Vol. 23, pp. 373-402, 1999.
9. L. O. Chua, and T. Roska: "The CNN Paradigm", *IEEE Trans. on Circuits and Systems*, 40:147-156, 1993.
10. T. Roska and L. O. Chua, "The CNN Universal Machine: An Analogic Array Computer", *IEEE Transactions on Circuits and Systems–II: Analog and Digital Signal Processing*, Vol. 40, pp. 163-173, March 1993.
11. T. Roska and L. Kék, "CNN Software Library (Templates and Algorithms), Version 7.2", *Analogical and Neural Computing Laboratory, Computer and Automation Research Institute, Hungarian Academy of Sciences (MTA SzTAKI)*, DNS-CADET-15, Budapest, 1998.
12. Morita M., Bortolozzi F., Facon J. and Sabourin R., „Morphological approach of handwritten word skew correction", *International Symposium on Computer Graphics, Image Processing and Vision*, Rio de Janeiro, Brazil, Oct. 1998.

## Appendix – The Utilized Templates

Linear, non-isotropic templates

**DPROP**: $A=\begin{bmatrix}0 & 1.75 & 0\\ 0 & 3 & 0\\ 0 & 0 & 0\end{bmatrix}, B=0, z=3.75$

**BOTEP**: $A=\begin{bmatrix}0 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 0\end{bmatrix}, B=\begin{bmatrix}-1 & -1 & -1\\ -1 & 1 & -1\\ 0 & 0 & 0\end{bmatrix}, z=-5.5$

**SKLHV1**: $A=\begin{bmatrix}0 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 0\end{bmatrix}, B=\begin{bmatrix}-1 & -1 & 0\\ -1 & 7 & 1\\ 0 & 1 & 1\end{bmatrix}, z=-2$

**SKLHV2** … **SKLHV8**: Same as **SKLHV1**, B must be rotated around the center element one by one

**CONN4SE**:

$$A=\begin{bmatrix}0 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 0\end{bmatrix}, B=\begin{bmatrix}0 & 0 & 0\\ 0 & 1 & 1\\ 0 & -1 & -1\end{bmatrix}, z=-3$$

**CONN4SW**, **CONN4NW**, **CONN4NE**: Same as **CONN4SE**, but B must be rotated around the center element by 90°, 180° and 270°

Linear, isotropic templates

| Template | Feedback (A) | | | Control (B) | | | Current | BCond |
|---|---|---|---|---|---|---|---|---|
| | $a_0$ | $a_1$ | $a_2$ | $b_0$ | $b_1$ | $b_2$ | Z | $B_c$ |
| **PRUNE** | 3 | 0.5 | 0 | 0 | 0 | 0 | -1.5 | 0 |
| **BPROP** | 3 | 0.25 | 0.25 | 0 | 0 | 0 | 3.75 | -1 |
| **WPROP** | 3 | 0.25 | 0.25 | 0 | 0 | 0 | -3.75 | 1 |
| **CPATCH** | 3 | 0.25 | 0.2 | 0 | 0 | 0 | 3.65 | -1 |
| **CWPATCH** | 3 | 0.25 | 0.2 | 0 | 0 | 0 | -3.65 | -1 |
| **GETEP** | 3 | 0.25 | 0.25 | 0 | 0 | 0 | -1.25 | -1 |

$$A=\begin{bmatrix}a_2 & a_1 & a_2\\ a_1 & a_0 & a_1\\ a_2 & a_1 & a_2\end{bmatrix}, B=\begin{bmatrix}b_2 & b_1 & b_2\\ b_1 & b_0 & b_1\\ b_2 & b_1 & b_2\end{bmatrix}, z$$