Segmentation-Free Printed Character Recognition by Relaxed Nearest Neighbor Learning of Windowed Operator

HAE YONG KIM

Universidade de São Paulo, Departamento de Engenharia Eletrônica,
Av. Prof. Luciano Gualberto, trav. 3, 158, São Paulo, SP, Brazil, CEP: 05508-900
hae@lps.usp.br, http://www.lps.usp.br/~hae/

Abstract. The segmentation is considered by many researchers as the key technology for a reliable optical character recognition (OCR) system. To accomplish a sound segmentation, many alternative techniques have been recently proposed. This paper presents a new technique to recognize characters without an explicit segmentation. It is based on the automatic construction of windowed operator by relaxed nearest neighbor learning. It has been implemented, tested and yielded excellent recognition accuracy and computational performance.

Keywords: Optical character recognition, nearest neighbor learning, windowed operator design, kd-tree

1 Introduction

Commercial machine-printed OCR systems are now being widely used as a reasonably reliable technology. Nevertheless, they are still far from reading as well as a human. Thus, the effort to increase their accuracy is still going on. Roughly, an OCR system can be divided into two main stages: segmentation and classification. The segmentation consists on splitting a scanned bitmap image of a document into individual characters and the classification, also called recognition, consists on categorizing each segmented character bitmap into a character class.

Among these two main constituents, the segmentation is considered by most OCR researchers as weaker component of a page reader, responsible for committing most errors. Note that the segmentation of real-world document images can not be performed on the basis of vertical white spaces only, for they are usually noisy, distorted by the scanning process and contain many touching characters. Thus, the segmentation has been subject of many investigations and many alternative techniques have been proposed to perform a reliable segmentation. Classic and unconventional segmentation techniques are surveyed in several papers [Casey and Lecolinet, 1996; Fujisawa et al., 1992; Nadler, 1984; Elliman and Lancaster, 1990]. One now widely used technique is the resegmentation of improperly-segmented characters detected during some posterior stage. In classic OCR systems, the result of the segmentation is passed on to the classifier and there is no feedback loop that permits the segmenter to make use of the knowledge gained at classification. In newer approaches, the improperly-segmented characters detected during classification is sent back to the segmenter and required to be resegmented. However, even this strategy can confound, for example, 'rn' with 'm' and 'cl' with 'd'. Thus, many recent OCR systems use also a language-specific contextual analysis to detect errors in both classification and segmentation, and these informations is feedbacked to earlier stages to correct errors. Another 'unconventional' technique is the segmentation and recognition of whole words instead of individual characters, specially practical for handwritten texts.

The segmentation-free character recognition, also called recognition-based segmentation, is one more alternative segmentation technique [Casey and Lecolinet, 1996]. It consists on simply bypassing the segmentation stage or, more precisely, segmenting the characters implicitly as a consequence of their recognition. The principal advantage of this approach is that the segmentation, the weakest component of an OCR system, becomes dispensable. Eliminating the segmentation, the feedback loop linking the classification back to the segmentation is also eliminated, and this may yield more elegant OCR programs.

Probably, the simplest segmentation-free technique is the template matching [Duda and Hart, 1973], also referred as peephole method [Mori et al., 1992], n-tuple feature [Ullmann, 1969; Jung et al., 1996] or hit-or-miss operator [Gonzalez and Woods, 1992]. It consists on choosing appropriate pixels for both black and white regions so that, looking through the selected peepholes or window, a given character is decided to belong to a class if it exactly matches the template. Some papers, as [Ullmann, 1969; Jung et al., 1996], have applied template matching in previously segmented characters to recognize their class, but this approach is not of our interest since it is not a segmentation-free technique. The template matching can also be applied in a completely unsegmented raw document bitmap. This operation can be conceived as if the template is copied on a transparency, then this transparency is superimposed onto bitmap to be recognized, and it is shifted to every possible positions of the bitmap, looking for locations where each pixel of template has the same color as in bitmap. In the exactly matching positions, a target character is supposed to be found. The main drawback of this approach is its fragility to recognize degraded and noisy characters from the real-world, for a single wrong pixel can lead to misclassification.

To face this problem, many template matchings should be joined by boolean operations 'or' in order to construct an intricate windowed operator that matches any ideal or degraded characters from one class and simultaneously rejects all shifted and corrupted characters from any other classes. By windowed operator (also called windowed filter or mask operation, and denoted in this paper as W-operator for short) we mean an image transformation that assigns to an output pixel a value that is function of the input image values at peepholes. As W-operator becomes more and more elaborate and complex, two other drawbacks of the template matchings joined by 'or' arise: the downfall of the computational performance and the difficulty to design manually such a complex W-operator. We are of the opinion that, in practice, this approach will never reach a good recognition accuracy, due to almost limitless quantity of informations necessary and consequent requirement of huge memory and processing time.

From the above discussion, two points become clear. First, that some kind of tool is necessary to help the user to project the W-operator. Second, that the W-operator produced by this tool has to be efficient, in terms of memory and processing time. Many different tools have been proposed in literature to aid the design of W-operator, for example, the use of the artificial intelligence [Schmitt, 1989] and fuzzy expert system [Kim et al., 1997]. Though these and analogous techniques can somehow help the user, they still are not a fully automated system and are not focused on obtaining an efficient W-operator.

As a different strategy, some works employ the training input-output sample images to project automatically a W-operator by a learning algorithm. [Harvey and Marshall, 1996] propose the use of genetic algorithm to search for the optimum gray-scale morphological filter that attenuate noise, using sample images to evaluate the fitness function of the projected filter. [Russo, 1996] proposes the use of neuro-fuzzy operator to filter noisy images. This filter, that combines the neural and fuzzy paradigms, is trained using a genetic algorithm and the fitness function is again evaluated by calculating the difference between the processed and noise-free images. [Barrera et al., 1997] present an algorithm, named ISI, to automatically project binary morphological operators using training sample images. In that paper, the operator is designed to mimic the behavior observed in the sample images. The patterns that are absent in sample images are considered as statistically unimportant and consequently labeled 'don't care'.

Undoubtedly, three characteristics ought to be taken into account in the analysis of a technique that projects W-operators by computational learning. The first is the extent of the W-operator class that the technique can project. The broader this class, the more flexible the technique is, for it can be used in a wider range of image processing applications. From this point of view, techniques specially aimed to a particular purpose (as [Harvey and Marshall, 1996; Russo, 1996] that are directed to attenuate noise) are inadequate to character recognition, for the W-operator generated by them will always belong to a subclass fit to their original purposes but may not be suitable in OCR. The second is the quality of the projected filter, that is, the expectation of similarity between the processed and ideal images. This quality mainly lies on the capability of generalizing the behavior of the W-operator to the patterns that were not present in the training sample images. From this point of view, the techniques that do not have an explicit generalization policy, as [Barrera et al., 1997], seems to be inapt to our purposes. And the third is the computational complexity of the algorithm, that is, the velocity of technique to project a W-operator and to apply it to an image. Analyzing the computational complexity of the algorithms presented in works [Harvey and Marshall, 1996; Russo, 1996; Barrera et al., 1997], seemingly they are very slow in the operator design stage and no substantial effort is made to speed up the operator application stage.

The technique that seems to fulfill these three desired characteristics, insofar as possible, is the relaxed nearest neighbor (NN) learning applied in the design of the W-operator [Kim, 97; Kim and Cipparrone, 1998]. First because, since it is not designed for any specific image processing purpose, it can be used with no restrictions in the character recognition. Though this technique can be used in gray-scale or color images, it yields specially good results for binary images. Secondly, it has a sound generalization strategy, namely the NN learning. To process a pattern that has no identical example in sample images, the most similar pattern is sought. Here, the distance between two binary patterns is computed by counting the number of non-matching pixels. The use of a generalization strategy greatly reduces the quantity of samples necessary to reach a good accuracy, managing to escape from the requirement of classifying almost all possible patterns as either 'target' or 'non-target'. The true NN searching is a computationally hard problem and apparently no fast solution is known for high dimensions. The kd-tree and similar tree-based data structures [Bentley, 1975; Friedman et al., 1977; Preparata and Shamos, 1985; Pearl, 1984] that are widely used to accelerate the searching problem, has their performance severely degraded in high dimensions. Thus, to fulfill the third desired characteristic, the NN searching was loosened, accepting that one of the near neighbors may be sometimes found, instead of coercing to find the true NN. This relaxed NN learning can be performed extremely fast, up to millions of times faster than the true NN searching, using a data structure that we have named cut-tree [Kim, 97].

We remark here that many OCR systems already use some kind of learning to train the classifier. The main difference of our strategy is that a single training makes possible the segmentation as well as the classification. Besides, the use of a fast data structure (cut-tree) makes viable the application of the function constructed by the learning in every possible positions of the image. This increases the reliability of segmentation and classification because the decision is made on a statistical basis. If a character occupies, say, 13×13 pixels in the scanned image, roughly 169 tests is made and the final decision is based on all these tests. This amount of tests is only possible in a current-day computer due to the use of cut-tree.

The relaxed NN learning has been implemented and tested in OCR. The proposed technique takes only some seconds in a present-day computer, not only in the recognition of characters in a page but also in the training. It has achieved 99.7% of recognition accuracy, to recognize characters of the same font and size as the training ones. Meanwhile, OmniPage 9.0, one of best selling OCR programs, recorded 97.2% of accuracy using the same bitmaps.

2 Character-recognizing binary W-operator design

Character printing is a binary operation by nature. A printed text has only two colors: the foreground (usually black) and the background (usually white). As a consequence, most OCR systems deal only with binary images. A digital binary image Q may be defined as either a function $Q : \mathcal{Z}^2 \rightarrow \{0,1\}$ (0 is usually defined as a black pixel and 1 as a white one) or as a subset $Q \subset \mathcal{Z}^2$ (a pixel is white iff it belongs to the subset). Since these two definitions are strictly equivalent, we adopted the first one. The support of a binary image Q is a finite subset of $\mathcal{Z}^2$ where the image is actually defined, and will be denoted as $S(Q)$. The size of support is the number of pixels of the image. Out of its support, an image is considered to be filled with a background color, usually white.

A binary image operator or binary image filter

$$\Psi : (\mathcal{Z}^2 \rightarrow \{0,1\}) \rightarrow (\mathcal{Z}^2 \rightarrow \{0,1\})$$

is a function that maps a binary image into another. The word 'filter' is somewhat confusing term. Different branches of image processing assigns different meanings to the same word. In our case, a transformation that rejects all but one class of characters can properly called filter, because a filter originally means a device for separating some substances from others. Nevertheless, to prevent misunderstandings, we will avoid to use this term, using rather the word 'operator'. The space of operators defined above is excessively large even considering that an image is defined only within its support, what obstructs the project and computational codification of an operator. Thus, in most cases, only those operators where the color of an output pixel is determined by a corresponding small area of the input image, called window or peepholes, are used. We will refer to them as windowed operators (W-operators). Formally, a W-operator Ψ is an operator that is defined via a sequence of points called window

$$\vec{W} = (W_1, ..., W_w),\ W_i \in \mathcal{Z}^2$$

and a characteristic function

$$\psi : \{0,1\}^w \rightarrow \{0,1\}$$

as follows:

$$\Psi(Q)(p) = \psi(Q(W_1 + p), ..., Q(W_w + p))$$

where $Q : \mathcal{Z}^2 \rightarrow \{0,1\}$ and $p \in \mathcal{Z}^2$.

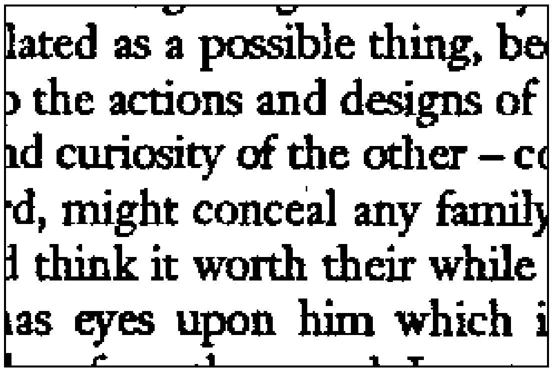
lated as a possible thing, be
the actions and designs of
d curiosity of the other – c
d, might conceal any famil
l think it worth their while
as eyes upon him which i

Figure 1: A portion of input sample (A^x)

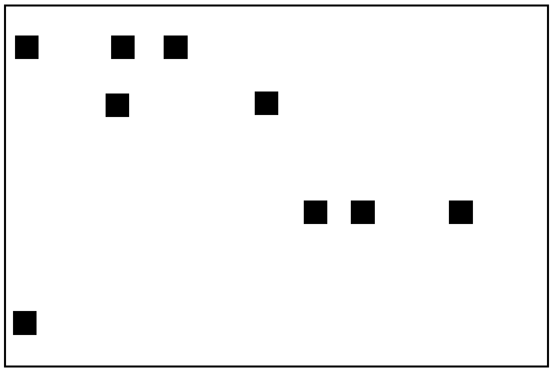

Figure 2: A portion of output sample (A^y)

Let the binary images $A^{\mathbf{x}}$, $A^{\mathbf{y}}$, $Q^{\mathbf{x}}$ and $Q^{\mathbf{y}}$ be respectively input sample, output sample, image to be recognized and ideal output image. We will suppose that there is only one pair of training images ($A^{\mathbf{x}}$ and $A^{\mathbf{y}}$), because if there are many sample images they can be 'glued' together to form a single pair. In our application, the input sample image is formed by 'pasting' a set of scanned document images and the output sample image is manually edited to indicate with a black rectangle the positions where the target characters are located (Figs. 1 and 2). Our goal is to project a W-operator Ψ, using sample images $A^{\mathbf{x}}$ and $A^{\mathbf{y}}$, such that when Ψ processes an image to be recognized $Q^{\mathbf{x}}$ (Fig. 3), a processed image $Q^{\mathbf{p}} = \Psi(Q^{\mathbf{x}})$ (Fig. 4) that indicates positions of target characters is produced. If the learning process was effective, this image may be expected to be 'similar' to an usually unknown ideal output image $Q^{\mathbf{y}}$ that perfectly indicates the positions of target characters.

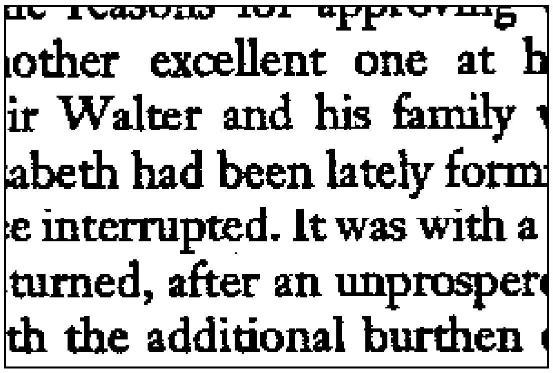
ne reasons for approving
other excellent one at h
ir Walter and his family
abeth had been lately form
e interrupted. It was with a
turned, after an unprosper
th the additional burthen

Figure 3: A portion of page to be recognized ($Q^{\mathbf{x}}$)

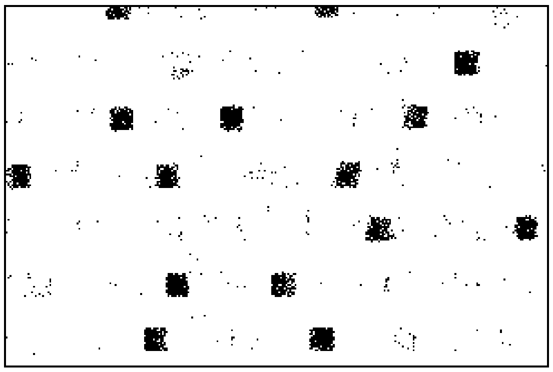

Figure 4: Output of W-operator ($Q^{\mathbf{p}}$)

Let m be the number of pixels of the sample images $A^{\mathbf{x}}$ and $A^{\mathbf{y}}$. The colors of pixels of $A^{\mathbf{x}}$, that belong to the window $\vec{W}$ translated to a pixel p, define a point

$$a^{\mathbf{x}} = [A^{\mathbf{x}}(W_1 + p), ..., A^{\mathbf{x}}(W_w + p)]$$

in the space $\{0,1\}^w$. This point, whose coordinates are the colors of pixels within the window $\vec{W}+p$, is called a sample pattern or template. To each template $a^{\mathbf{x}}$ so obtained, there is an associated output color or value $a^{\mathbf{y}} = A^{\mathbf{y}}(p) \in \{0,1\}$. The sample pattern and its output value together form an example or sample. Let us denote the data obtained when all pixels of $A^{\mathbf{x}}$ and $A^{\mathbf{y}}$ are scanned as

$$\vec{a} = (a_1, ..., a_m) = ((a_1^{\mathbf{x}}, a_1^{\mathbf{y}}), ..., (a_m^{\mathbf{x}}, a_m^{\mathbf{y}}))$$

and call it sample sequence.

In the W-operator designing stage, a learning algorithm

$$\mathbf{A} : \left(\bigcup_{m \geq 1} (\{0,1\}^w \times \{0,1\})^m \right) \rightarrow (\{0,1\}^w \rightarrow \{0,1\})$$

receives a sample sequence $\vec{a}$ and generates a characteristic function $\psi = \mathbf{A}(\vec{a})$. The function ψ and the window $\vec{W}$ together represent the W-operator Ψ.

Let us suppose that an unknown joint probability distribution P in the space $\{0,1\}^w \times \{0,1\}$ had generated independently each element of the sequence $\vec{a}$. If the resolution of the images are fine enough, the window size is sufficiently large and the images are noiseless, it can be expected that each input pattern is associated with an unique output color, that is, $P(a_j^{\mathbf{x}}, 0)$ is either 0 or 1, and $P(a_j^{\mathbf{x}}, 1) = 1 - P(a_j^{\mathbf{x}}, 0)$. In practice, the suppositions above do not hold perfectly and thus $P(a_j^{\mathbf{x}}, 0)$ may assume values different from 0 and 1. This makes possible that conflicting examples appear in a sample sequence. We say two examples $(a_i^{\mathbf{x}}, a_i^{\mathbf{y}})$ and $(a_j^{\mathbf{x}}, a_j^{\mathbf{y}})$ conflict if $a_i^{\mathbf{x}} = a_j^{\mathbf{x}}$ but $a_i^{\mathbf{y}} \neq a_j^{\mathbf{y}}$.

Let us suppose also that the same probability distribution P generates independently each element of the sequence to be recognized

$$\vec{q} = (q_1, ..., q_n) = ((q_1^{\mathbf{x}}, q_1^{\mathbf{y}}), ..., (q_n^{\mathbf{x}}, q_n^{\mathbf{y}}))$$

that forms the image to be recognized $Q^{\mathbf{x}}$ and ideal output image $Q^{\mathbf{y}}$ (n is the number of pixels of the images $Q^{\mathbf{x}}$ and $Q^{\mathbf{y}}$). In the W-operator application stage, the characteristic function ψ is applied to each input query pattern $q_i^{\mathbf{x}}$, generating n output processed colors or values $q_i^{\mathbf{p}} = \psi(q_i^{\mathbf{x}})$. The sequence of n processed colors forms the processed image $Q^{\mathbf{p}}$.

If $\mathbf{A}$ was a good learning algorithm, what value should $q_i^{\mathbf{p}} = \psi(q_i^{\mathbf{x}}) = \mathbf{A}(\vec{a})(q_i^{\mathbf{x}})$ be? Intuitively, given a query pattern $q_i^{\mathbf{x}}$, we should search for a training sample $(a_j^{\mathbf{x}}, a_j^{\mathbf{y}}) \in \vec{a}$ whose sample template is the same as the query pattern ($a_j^{\mathbf{x}} = q_i^{\mathbf{x}}$). Then the sample output color $a_j^{\mathbf{y}}$ should be chosen as the processed value, that is, $q_i^{\mathbf{p}} = \psi(q_i^{\mathbf{x}}) = a_j^{\mathbf{y}}$. Now suppose that, for a given query pattern $q_i^{\mathbf{x}}$, there are more than one matching template, say, N samples

$$(a_{j_1}^{\mathbf{x}}, a_{j_1}^{\mathbf{y}}), ..., (a_{j_N}^{\mathbf{x}}, a_{j_N}^{\mathbf{y}})$$

where $a^{\mathbf{x}}_{j_k} = q^{\mathbf{x}}_i$, $1 \leq k \leq N$. As has been discussed above, in an ideal environment $P(a^{\mathbf{x}}_j, 0)$ is either 0 or 1, and consequently N output colors will be the same, either black or white and this color should be assigned to $\psi(q^{\mathbf{x}}_i)$. But in view of the distortion in the scanning process, human errors in editing the sample output image etc., there can be conflicting samples. If one or more of N output colors are different from the others, we should assign $\psi(q^{\mathbf{x}}_i) = \text{mode}(a^{\mathbf{y}}_{j_1}, ..., a^{\mathbf{y}}_{j_N})$. Mode is the value that occurs most frequently in a given sequence.

The strategy described above seems to be very natural and intuitive. But it is useless in practice, for no policy has been defined to attribute an output value to query patterns that have no exactly matching sample template. Moreover, in practice most of query patterns do not have an exactly matching template. To verify this claim, consider first that the window to be used in OCR can not be too small, because such a choice, despite speeding up the processing, may produce many conflicting samples. Thus, in our application we have used two windows with 35 and 37 peepholes and we deem that significantly smaller windows can not be used to recognize characters from a real-world document. Assuming $w = 35$, the size of the pattern space is 2^{35} and a characteristic function $\psi : \{0, 1\}^{35} \rightarrow \{0, 1\}$ has to be designed. To achieve this goal without a generalization policy, a sample sequence of length at least 2^{35} would have been necessary, or 9000 pairs of 2000×2000 binary sample images! Evidently, such a demand for sample images is not possible to be fulfilled. To escape from the exigency of classifying almost all possible patterns as either 'target' or 'non-target', a heuristic generalization is necessary. By heuristic generalization we mean a strategy to extend the behavior of the W-operator to unprogrammed patterns, based only on small number of catalogued prototype templates. Using the nearest neighbor (NN) as generalization strategy, a good recognition accuracy was obtained using only one pair of 2424×818 sample images.

Supposing that each character class corresponds to a set of continuous regions in the pattern space, the NN learning seems to be a reliable generalization scheme to be used in W-operator learning [Kim and Cipparrone, 1998], because two similar patterns is likely to belong to a same class. The NN strategy can be defined as follows: for a given query pattern $q^{\mathbf{x}}_i$, its processed color $q^{\mathbf{P}}_i = \psi(q^{\mathbf{x}}_i) = \mathbf{A}(\vec{a})(q^{\mathbf{x}}_i)$ is the mode among the output values of the NNs of $q^{\mathbf{x}}_i$ in the training sequence.

3 Nearest neighbor searching algorithms

Using the NN learning, a pattern that has a similar sample template is correctly classified, even if there is no exactly matching template. To use the NN learning in practice, a good algorithm for the NN searching problem is required. The NN searching problem can be stated as follows: "Given m training points $a^{\mathbf{x}}_1$, ..., $a^{\mathbf{x}}_m$ and n query points $q^{\mathbf{x}}_1, ..., q^{\mathbf{x}}_n$ (all points in the space $\{0, 1\}^w$) find, for each query point $q^{\mathbf{x}}_i$, the nearest training point (or points)." Note that there can be two or more equally distant nearest points. In OCR, the points are binary patterns and the distance between two patterns is the number of non-matching pixels.

The NN searching problem has a trivial and naive solution. In this solution, the distances between each $q^{\mathbf{x}}_i$ and all m training points have to be evaluated and hence the nearest training points are chosen. Henceforth we will refer to this as brute-force algorithm. Since the distances between all m training points and all n query points have to be computed, and as each distance evaluation demands $O(w)$ operations, the brute-force's computational complexity is $O(wmn)$. This analysis proves that this algorithm is excessively slow. A practical experiment shows that a present-day computer would take around one year to execute the application depicted in this paper (after one-day processing, only 0.3% of a page had been processed)!

There are faster algorithms for the NN searching. One such algorithm is the Voronoi diagram. Despite its appealing properties, the Voronoi diagram can be used in practice only for two-dimensional problems, because the quantity of vertices and edges of Voronoi diagram grows exponentially as the dimension increases [Preparata and Shamos, 1985], requiring a proportional quantity of memory.

Another widely used algorithm is the kd-tree. The kd-tree is a generalization of the simple binary tree used for sorting and searching and it provides an efficient searching mechanism for examining only those points close to the query point, thereby reducing the computational effort. The root of kd-tree represents the set of all training points. Each non-terminal node has two successor nodes that represent two subsets defined by partitioning parent's training points. The terminal nodes represent mutually exclusive small subsets of training points. The kd-tree data structure enables to find quickly a terminal node that contains a set of training points close to the query point $q^{\mathbf{x}}_i$. Then, only the distances between the training points within this node and the query point have to be evaluated to compute a candidate point to the NN. A recursive backtracking process is necessary to certify that the candidate is indeed the NN. For a thorough exposition, the reader is referred to [Bentley, 1975; Friedman et al., 1977; Preparata and Shamos, 1985; Pearl, 1984].

The kd-tree has been implemented and tested in OCR application. To use the kd-tree in the OCR, the output colors $a^{\mathbf{y}}_j$ have to be stored in the terminal nodes, in addition to the corresponding input patterns $a^{\mathbf{x}}_j$. In practice, to minimize the use of the memory, only the coordinates (line and column numbers) of the pattern $a^{\mathbf{x}}_j$ in $A^{\mathbf{x}}$ were stored in the terminal nodes, instead of input patterns. Note that the same coordinates indicate the location of output color $a^{\mathbf{y}}_j$

in $A^{\mathbf{y}}$. These informations, together with the window $\vec{W}$ and images $A^{\mathbf{x}}$ and $A^{\mathbf{y}}$, allow to evaluate $a_j^{\mathbf{x}}$ and $a_j^{\mathbf{y}}$. A kd-tree, constructed as above, can be viewed as a characteristic function ψ. Given a pattern $q_i^{\mathbf{x}}$, a searching in the kd-tree finds the most similar training patterns. The mode of the nearest patterns' output colors is then defined as the processed output value $q_i^{\mathbf{P}}$.

According to [Preparata and Shamos, 1985], kd-tree uses $O(wm)$ storage, a fair quantity. The construction of a kd-tree can be effected in time $O(wm \log m)$, that is, the construction is extremely fast. Nevertheless, the n points searching in a kd-tree takes $O(nwm^{(1-1/w)})$, meaning that the computational performance of the searching in the kd-tree degrades quickly as the dimension of the window's size increases, soon becoming as bad as brute-force. In practice, the kd-tree that recognizes a character could be constructed in a few seconds. But kd-tree processed only 0.9% of a page in one day, using a window of size 35. That is, the kd-tree would take four months to process a whole page. This is three times faster than the brute-force, but unfortunately still is not helpful...

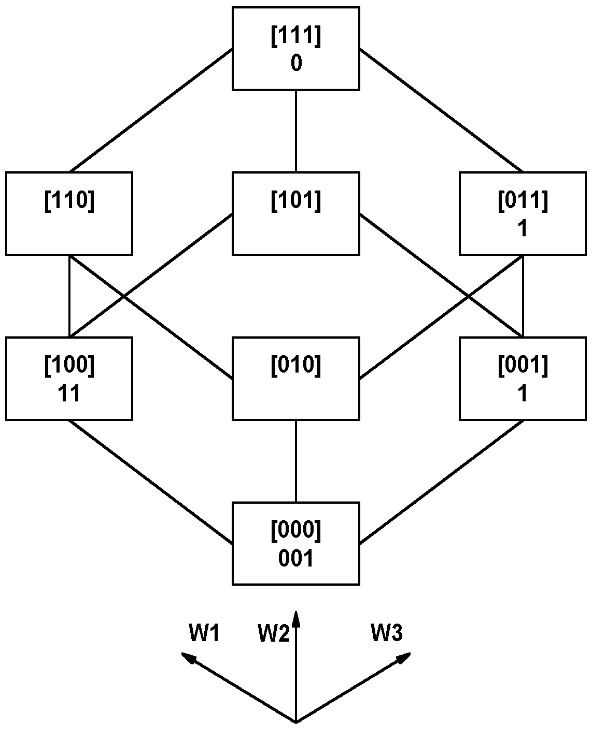

Figure 5: Pattern space

4 Relaxed nearest neighbor learning

An alternative technique can be obtained by substituting the requirement of finding the NNs for finding one near neighbor. The kd-tree was modified for this relaxed searching and the modified version was named cut-tree [Kim, 1997]. In essence, the cut-tree is obtained from the kd-tree by not allowing the recursive backtracking process in the searching algorithm. Besides increasing the speed, this alteration also eliminates the necessity of storing input patterns $a_j^{\mathbf{x}}$ in the terminal nodes. That is, the line and column numbers of $a_j^{\mathbf{x}}$ in $A^{\mathbf{x}}$, as well as the images $A^{\mathbf{x}}$ and $A^{\mathbf{y}}$ themselves, are no more stored. Only the output color $a_j^{\mathbf{y}}$ is stored, diminishing the memory use. This leads to a $O(m)$ storage that does no more depend on the window's size w (the analysis is shown in the final part of this section). The cut-tree, at most times, may not find the true NN, but only one 'near neighbor'. On the other hand, this approximated NN searching can be executed millions times faster than the strict searching. To expose the construction of kd-tree, let be given m sample template points with the corresponding output colors:

$$\vec{a} = (a_1, ..., a_m) = ((a_1^{\mathbf{x}}, a_1^{\mathbf{y}}), ..., (a_m^{\mathbf{x}}, a_m^{\mathbf{y}})),$$

where $a_j^{\mathbf{x}} \in \{0,1\}^w$ and $a_j^{\mathbf{y}} \in \{0,1\}$. In the cut-tree generating process, the pattern space $\{0,1\}^w$ is split into two halves, and all sample patterns with black color in the splitting axis W_s will belong to one half-space and those with white color to another. The dimension of half-spaces obtained is one less than that of original space, that is, $\{0,1\}^{w-1}$. To obtain an optimized tree, the splitting axis $s \in [1...w]$ has to be chosen so that the resulting two half-spaces contain as equal as possible number of sample points. If the difference of points is always zero or one, a perfectly balanced tree will be created. As the difference of number of samples in two halves increases, more and more degenerated tree will be constructed. For each one of two half-spaces obtained, the splitting process continues recursively, generating smaller and smaller spaces. This process stops when each space contains only either samples with a same output color or only samples with a same input pattern but with two different output colors. In the first case, a terminal node is created and the output color is stored in it. In the second case, called conflict, the mode of the output colors is evaluated and stored. Each splitting corresponds to an internal node of the cut-tree and each final sub-space corresponds to a terminal node.

For example, let $w = 3$ and consider the following training sample sequence:

$$\begin{aligned} \vec{a} = (\quad & [000,0], [000,0], [000,1], [001,1], \\ & [100,1], [100,1], [011,1], [111,0] \quad). \end{aligned}$$

The notation $[011,1]$ indicates an training example where the input pattern is black in axis W_1, white in axes W_2 and

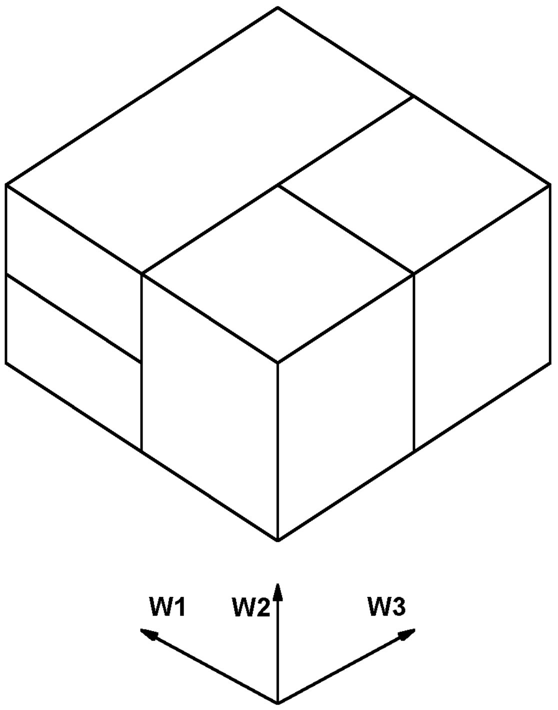

Figure 6: Partition of pattern space

W_3, and the output color is white. As the pattern space is three-dimensional, it can be represented as a cube (Fig. 5). Let us denote the space $\{0,1\}^w$ as [XXX], since a point in this space can vary its value in any of three axes. To indicate that a sub-space has a fixed value in some axis, the corresponding X may be substituted by 0 or 1.

If the space [XXX] is split at axis W_2, the two resulting half-spaces will contain 6 and 2 samples, while if the splitting occurs at axes W_1 or W_3, two half-spaces will contain 5 and 3 samples. So, either W_1 or W_3 may be chosen as splitting axis, but not W_2. Supposing that W_1 is chosen, the two resulting half-spaces are [0XX] and [1XX]. This and successive cleavages are illustrated in Fig. 6 and the cut-tree produced in Fig. 7.

The half-space [0XX] contains 5 samples

$$([000,0],[000,0],[000,1],[001,1],[011,1])$$

and the half-space [1XX] contains 3 samples

$$([100,1],[100,1],[111,0]).$$

Splitting the half-space [0XX] at axis W_2, the two resulting quarter-spaces will contain 4 and 1 samples, while splitting at W_3 two quarter-spaces with 3 and 2 samples will be yielded. Thus, W_3 must be chosen to split this half-space, generating two quarter-spaces: [0X0] with samples $([000,0],[000,0],[000,1])$ and [0X1] with samples $([001,1],[011,1])$.

The quarter-space [0X0] is not split again since it contains three examples with a same pattern. As the outputs of three examples are not a unique value, there is a conflict. To solve it, the mode is evaluated and hence 0 is chosen as the output color. The quarter-space [0X1] also is not split since its two samples have a same output color 1.

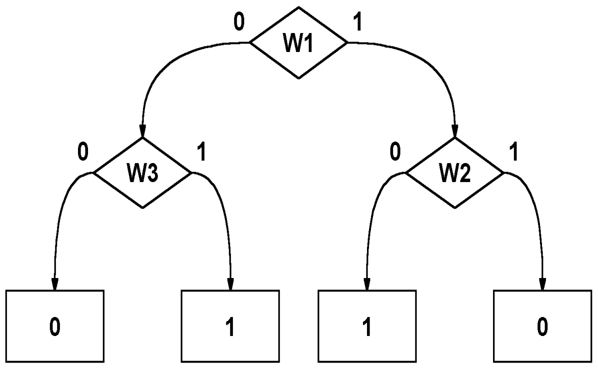

Figure 7: Cut-tree

The half-space [1XX] can be split at axis W_2 or W_3. No matter what axis is chosen, the resulting quarter-spaces will contain 2 and 1 samples. Let us assume that the splitting has occurred at W_2. Then the quarter-space [10X] will contain samples $([100,1],[100,1])$ and the quarter-space [11X] will contain the sample $[111,0]$. These quarter-spaces are not split again as in each there is only one output color.

A cut-tree represents a characteristic function ψ. The value of the function $q_i^{\mathrm{P}} = \psi(q_i^{\mathrm{x}})$ is evaluated by performing a searching in the cut-tree. For example, consider the cut-tree of Fig. 7 and suppose that $\psi(q_i^{\mathrm{x}})$ ought to be evaluated, where $q_i^{\mathrm{x}} = [101]$. In the root node, we are forced to choose the right sub-tree, for the query point has value 1 in axis W_1. In the node labeled W_2, we are conducted to the left because the query point has value 0 in axis W_2, reaching the external node with value 1 and therefore $\psi([101]) = 1$.

The construction algorithms of the cut-tree and kd-tree are quite similar. Therefore, a cut-tree can be built in average time $O(wm \log m)$, like kd-tree. If a cut-tree is built from m samples and if every terminal node contains only one sample, it is easy to see that the height of the cut-tree will be $\lceil \log_2 m \rceil + 1$ (where $\lceil \cdot \rceil$ stands for round up). Thus, if there can exist terminal nodes with more than one sample, the height will be at most $\lceil \log_2 m \rceil + 1$. Consequently, the searching of n query points in a cut-tree can be performed in time $O(n \log m)$, for the searching processes in a cut-tree and in a standard binary tree are identical. Note that

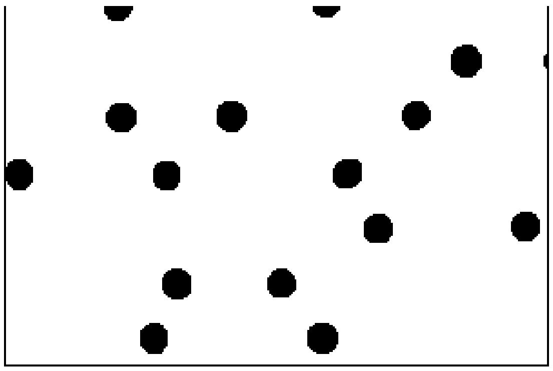

Figure 8: Filtering and thresholding

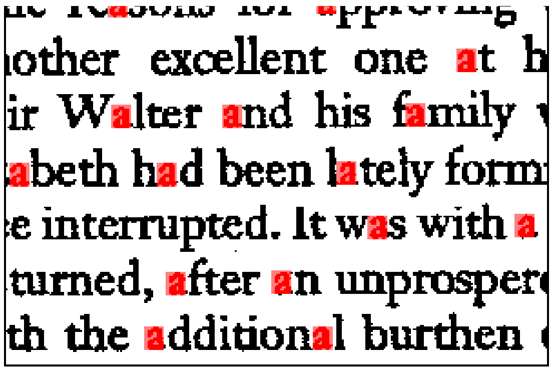
other excellent one at
ir Walter and his family
abeth had been lately form
e interrupted. It was with a
turned, after an unprosper
th the additional burthen

Figure 9: Recognition of ‘a’

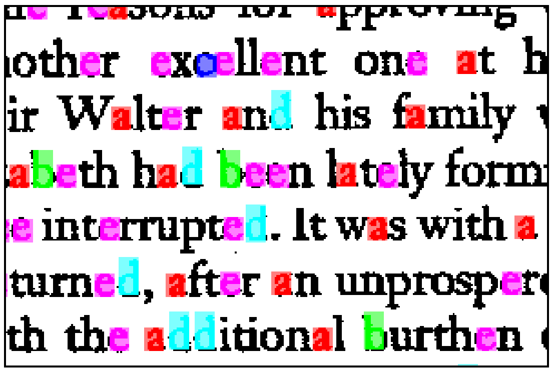
other excellent one at
ir Walter and his family
abeth had been lately form
e interrupted. It was with a
turned, after an unprosper
th the additional burthen

Figure 10: Recognition of ‘a’, ..., ‘e’

the searching complexity does not depend on the dimension w of the pattern space at all. Finally, if every terminal node contains only one sample, the number of terminal nodes will be m and the number of internal nodes will be $m-1$. As the quantity of informations stored in a terminal or internal node does not depend on any parameter, the total storage is simply $O(m)$. In practice, the total number of nodes is much smaller than $2m-1$, as many terminal nodes contain much more than one sample.

This analysis shows that both the construction and the searching are extremely fast in the cut-tree and the memory use is economical, even in high dimension. Experimental results confirm our analysis. As has been previously mentioned, a same OCR task takes one year using brute-force, four months using kd-tree and only 5 seconds using cut-tree. Seemingly, the cut-tree is far faster than other techniques found in the literature for the automatic filter designing [Harvey and Marshall, 1996; Russo, 1996; Barrera et al., 1997]. This claim can be argued by analyzing the complexity of the algorithms presented in those works.

5 Experimental results and discussions

The exposed technique has been used to recognize characters ‘a’, ..., ‘e’ in 6 low-quality and in 4 high-quality scanned pages.

The low-quality bitmaps were obtained from a poor-printed pocket book. A hand-held scanner was used to scan pages in 256 gray-scales at 600 dpi. A hand-held scanned usually generates more pattern mutations than a desktop model. The images were resampled at 200 dpi and thresholded, resulting 1212×818 binary images. Two such pages were ‘glued’ together and used as the input sample. Fig. 1 depicts a portion of this image. The output sample was edited manually, imprinting target characters with a rectangular mark (Fig. 2). A window with 35 peepholes scattered in 13×13 grid was used to recognize characters ‘a’, ‘c’, ‘e’ and another window with 37 peepholes scattered in 21×13 grid to recognize ‘b’ and ‘d’.

Processing Fig. 3 with W-operator designed to recognize ‘a’, Fig. 4 is yielded. Filtering this image with a 13×13 (or 21×13) linear filter and thresholding the result, Fig. 8 is obtained. The center of each connected component was detected and dilated up to the size of the character and the outcome was superimposed on the original image, resulting Fig. 9. Fig. 10 shows this technique repeated to characters ‘a’, ..., ‘e’. This technique has committed only 6 errors in 1860 recognized characters (99.7% of accuracy, Tab. 1). Meanwhile, OmniPage Pro 9.0, one of best selling commercial OCR programs, made 53 mistakes using the same images (97.2% of accuracy). Three out of six errors committed by our technique were in recognition of the character ‘b’, since sample images contained only 54 examples of characters ‘b’. The training took 56s and the application 5s in a Pentium-300. The W-operators that recognize ‘a’, ‘b’, ‘c’, ‘d’ and ‘e’ occupied respectively 487, 317, 271, 431 and 560 Kbytes.

The high-quality bitmaps were obtained by scanning deskjet-printed documents (Times New Roman 12pt, ‘A4’ paper) with a desktop scanner. The documents were initially scanned in 256 gray-scales at 600 dpi and then resampled to 200 dpi and thresholded, resulting 2082×1424 binary images. In order to obtain a sample page with approximately equal number of occurrences of each character, a document containing only lowercase characters 'a', ..., 'z' and blank spaces was generated randomly, printed and scanned like other documents. The same recognition process used for low-quality documents was used here and committed no error. OmniPage Pro 9.0 made one mistake in recognition of the same images.

	Relaxed NN	OmniPage Pro 9.0
$a \to ?$	0	15
$? \to a$	0	1
$b \to ?$	3	2
$? \to b$	0	1
$c \to ?$	1	2
$? \to c$	0	4
$d \to ?$	1	3
$? \to d$	0	8
$e \to ?$	1	16
$? \to e$	0	1
Total	6	53
Accuracy	99.7%	97.2%

Table 1: Error rate for low-quality bitmaps

As a future work, an omni-font OCR based on the relaxed NN learning is expected to be brought into being using a feature vector similar to that described in [Bokser, 1992]. This feature vector consists primarily of a blurred gray-scale reduction of the image, obtained by dividing the input image into zones and assigning to each a value related to the relative density of black pixels in that zone. Another future improvement is to construct by relaxed NN learning a cut-tree that recognizes all characters simultaneously, assigning to each character a different gray-scale.

6 Conclusions

This paper has presented a new segmentation-free OCR technique. It is based on automatic design of windowed operator. The use of nearest neighbor learning has been considered first but it is computationally too hard problem. Thus, an approximate solution offered by cut-tree has been proposed to be used. This paper has also presented experimental informations that confirms the effectiveness of the proposed technique.

7 References

1. [Barrera et al., 1997] Barrera, J., Dougherty, E. R. and Tomita, N. S.: 'Automatic programming of binary morphological machines by design of statistically optimal operators in the context of computational learning theory', *J. Electronic Imaging,* vol. 6, no. 1, pp. 54-67, 1997.

2. [Bentley, 1975] Bentley, J. L.: 'Multidimensional binary search trees used for associative searching', *Comm. ACM,* vol. 18, no. 9, pp. 509-517, 1975.

3. [Bokser, 1992] Bokser, M.: 'Omnidocument technologies', *Proc. IEEE,* vol. 80, no. 7, pp. 1066-1078, 1992.

4. [Casey and Lecolinet, 1996] Casey, R. G. and Lecolinet, E.: 'A survey of methods and strategies in character segmentation', *IEEE Trans. Patt. Anal. Mac. Intell.,* vol. 18, no. 7, pp. 690-706, 1996.

5. [Duda and Hart, 1973] Duda, R. O. and Hart, P. E.: 'Pattern Classification and Scene Analysis', Wiley, 1973.

6. [Elliman and Lancaster, 1990] Elliman, D. G. and Lancaster, I. T.: 'A review of segmentation and contextual analysis techniques for text recognition', *Pattern Recognition,* vol. 23, no. 3/4, pp. 337-346, 1990.

7. [Friedman et al., 1977] Friedman, J. H., Bentley, J. L. and Finkel, R. A.: 'An algorithm for finding best matches in logarithmic expected time', *ACM T. Math. Software,* vol. 3, no. 3, pp. 209-226, 1977.

8. [Fujisawa et al., 1992] Fujisawa, H., Nakano, Y. and Kurino, K.: 'Segmentation methods for character recognition: from segmentation to document structure analysis', *Proc. IEEE,* vol. 80, no. 7, pp. 1079-1092, 1992

9. [Gonzalez and Woods, 1992] Gonzalez, R. C. and Woods, R. E.: 'Digital Image Processing', Addison-Wesley, 1992.

10. [Harvey and Marshall, 1996] Harvey, N. R. and Marshall, S.: 'The use of genetic algorithms in morphological filter design', *Signal Processing: Image Comm.,* 8, pp. 55-71, 1996.

11. [Jung et al., 1996] Jung, D. M., Krishnamoorthy, M. S., Nagy, G. and Shapira, A.: 'N-tuple features for OCR revisited', *IEEE Trans. Patt. Anal. Mac. Intell.,* vol. 18, no. 7, pp. 734-745, 1996.

12. [Kim and Cipparrone, 1998] Kim, H. Y. and Cipparrone, F. A. M.: 'Automatic design of nonlinear filters by nearest neighbor learning', *in Proc. IEEE Int. Conf. on Image Proc.* (Chicago), vol. 2, pp. 737-741, 1998.

13. [Kim et al., 1997] Kim, H. Y., Cipparrone, F. A. M. and Andrade, M. T. C.: 'Technique for constructing gray-scale morphological operators using fuzzy expert system', *Electron. Lett.,* vol. 33, no. 22, pp. 1859-1861, 1997.

14. [Kim, 1997] Kim, H. Y.: 'Quick construction of efficient morphological operators by computational learning', *Electron. Lett.,* vol. 33, no. 4, pp. 286-287, 1997.

15. [Mori et al., 1992] Mori, S., Suen, C. Y. and Yamamoto, K.: 'Historical review of OCR research and development', *Proc. IEEE,* vol. 80, no. 7, pp. 1029-1058, 1992.

16. [Nadler, 1984] Nadler, M.: 'A survey of document segmentation and coding techniques', *Comp. Vision, Graphics, and Image Processing,* vol. 28, pp. 240-262, 1984.

17. [Pearl, 1984] Pearl, J.: 'Heuristics: Intelligent Search Strategies for Computer Problem Solving', Addison-Wesley, 1984.

18. [Preparata and Shamos, 1985] Preparata, F. P. and Shamos, M. I.: 'Computational Geometry, an Introduction', Springer-Verlag, New York, 1985.

19. [Russo, 1996] Russo, F.: 'Nonlinear filtering of noisy images using neuro-fuzzy operators', *in Proc. IEEE Int. Conf. Image Proc.* (Santa Barbara, USA), vol. 3, pp. 412- 415, 1997.

20. [Schmitt, 1989] Schmitt, M.: 'Mathematical morphology and artificial intelligence: an automatic programming system', *Signal Processing,* vol. 16, no. 4, pp. 389- 401, 1989.

21. [Ullmann, 1969] Ullmann, J. R.: 'Experiments with the n-tuple method of pattern recognition', *IEEE. Trans. Computers,* vol. 18, no. 12, pp. 1135-1137, 1969.