A modified self-organizing neural net for shape extraction

Amitava Datta [a], Tamaltaru Pal [b], S.K. Parui [b,*]

[a] *Computer and Statistical Service Centre, Indian Statistical Institute, Calcutta, India*
[b] *Computer Vision and Pattern Recognition Unit, Indian Statistical Institute, 203 B.T. Road, Calcutta 700 035, India*

Received 14 February 1995; accepted 30 September 1995

Abstract

Some modifications on Kohonen's self-organizing feature map are discussed to make it suitable for finding skeletons of binary images. In Kohonen's feature map, the set of processors and their neighbourhoods are fixed and do not change in the learning process. This may pose problems when the set of input vectors represents a prominent shape. The reference vectors or weight vectors lying in zero-density areas are affected by input vectors from all the surrounding parts of the non-zero distribution [5]. Hence a shape extraction problem requires a dynamic change in the network topology. In the present paper, to overcome the limitations of Kohonen's feature maps, we propose a mechanism in which the set of processors and their neighbourhoods change adaptively during learning, to extract the shape of a binary object in the form of a skeleton.

Keywords: Neural net; Self-organization; Adaptive learning; Binary image; Character patterns; Skeleton

1. Introduction

Artificial neural network models have been of great interest for some years in various areas like optimization, pattern recognition, computer vision and image processing [2,7,8,12]. Several neural network models have been proposed so far for various tasks. Kohonen's self-organizing feature maps [6] are a particular type of neural network model. In Kohonen's feature mapping algorithm, a set of processors, having some initial

weights, is considered and the same input is fed to all of them. The best matching processor (the winner processor) is found and the weight vectors of this processor and its topological neighbours are adjusted accordingly. The process is repeated for all the inputs and several iterations are performed until the weights converge. The learning (adjustments) is unsupervised.

In Kohonen's self-organizing feature maps, the set of processors and their neighbourhoods are fixed and do not change during learning. This may pose problems in many situations. For example, when the set of input vectors represents a prominent shape, it may easily happen that the reference vectors or weight vectors lying in zero-density areas are affected by input vectors from all the surrounding parts of the non-zero distribution [5]. This is particularly true for a problem like shape extraction which demands dynamic change in network topology. In the present paper, we propose, to overcome the limitations of Kohonen's feature maps, a mechanism in which the set of processors and their neighbourhoods change adaptively during learning, in order to extract the shape of a binary object in the form of a skeleton.

Describing structure and shape of objects is sometimes necessary in image processing, pattern recognition and computer vision [1,10,11]. Character recognition systems often require skeletons for extracting the basic geometrical and topological features of the character patterns and hence, in the present work, such patterns have been paid particular attention. Other advantages of skeletons are to reduce the memory space required to store the essential structural information of the pattern and to simplify the data structure required for processing the pattern.

In the neural networks model discussed here, the weight updation rules are similar to those in Kohonen's self-organizing feature map. But during learning process new processors can be added to or old processors can be deleted from the network. The initial topology here is simple (linear with a small number of processors) which grows adaptively into a more complex topology to deal with complicated structures. Similar types of dynamic neural networks have been suggested by several authors for vector quantization [3], estimation of probability distributions in the plane [4], and shape classification [13].

In his model Kohonen has used the term array of processors to represent the network. Since in our model the processors are inserted/deleted during the learning process, we use the term list of processors to represent the net structure. In the list, each node represents a processor and the links represent the connections between processors. This makes the insertion/deletion of processors meaningful and we can follow the standard insertion/deletion algorithms available in the list processing literature. In our discussion, by net structure we mean a (linked) list of processors having linear or nonlinear structure.

In the next section we have four subsections. Section 2.1 describes the technique for simple patterns like arcs. Section 2.2 explains how to deal with more complicated patterns having branchings, forks and crossings (for convenience we call them together tree-like patterns). Section 2.3 considers patterns that contain loop structures. In all the cases the starting structure of the network is linear only. It is the dynamic nature of the net that enables it to learn the structure of the input pattern and to expand (topologically) accordingly.

2. The network and updation rule

Kohonen's feature mapping net is an array of processors where each processor is connected to one or more surrounding processors and every processor stores a weight vector. The map is adapted on the basis of a set of input feature vectors. The net of processors is normally representable in either one or two dimensions. Dimension of the weight vectors is the same as that of the input vectors which can be arbitrary. Suppose the array of processors under consideration is represented as $\{\pi_1, \pi_2, \ldots, \pi_n\}$. The *neighbourhood* N_i of a processor π_i is $\{\pi_p \mid \pi_p$ is connected to $\pi_i\}$ which includes π_i. The processor π_i has a weight vector W_i whose dimension is, say m. Suppose the set of input vectors is $S = \{P_1, P_2, \ldots, P_N\}$ where the dimension of each P_j is also m. Updation rule for the weight vectors is as follows.

Suppose at time instance t, P_j is presented to the net and let $W_i(t) = (w_{i1}(t), w_{i2}(t), \ldots, w_{im}(t))$ be the nearest weight vector to P_j. Then, the weight vectors of the processors within the neighbourhood of π_i are updated in the following way:

$$W_p(t+1) = W_p(t) + \alpha(t)\big[P_j - W_p(t)\big] \quad \text{for } \pi_p \in N_i \tag{1}$$

where $\alpha(t)$ is the gain term satisfying certain conditions [6].

2.1. Simple arc patterns

We shall first consider input patterns having an arc shape (e.g., character patterns '*S*', '*C*', '*N*', '*L*' etc.). The structure of these patterns is such that they can be represented by linear structures. Shape extraction from such arc shaped patterns consisting of planar points has been discussed in our earlier work [9]. The present discussion deals with binary images instead of planar point sets. We start with a net having a linear structure which is represented by a list of processors $[\pi_1, \pi_2, \ldots, \pi_n]$ where π_i is connected to exactly two processors π_{i-1} and π_{i+1} (the two end-processors are connected to exactly one processor each). Here the input feature vectors are 2-dimensional coordinates of object pixels and hence $m = 2$.

Here $S = \{P_1, P_2, \ldots, P_N\}$ is a set of N object pixels representing a binary image where $P_j = (x_j, y_j)$. On the basis of the pixels in S, the weight vectors of the processors π_i will be updated iteratively. Let the initial weight vectors of π_i be $(w_{i1}(0), w_{i2}(0))$. Suppose the pixel P_j is presented at the tth iteration. Let

$$\operatorname{dist}\big(P_j, W_k(t)\big) = \min_i \big[\operatorname{dist}\big(P_j, W_i(t)\big)\big]$$

where $W_i(t)$ is the ith weight vector at the tth iteration.
That is, π_k is the nearest processor from the pixel P_j. P_j updates the weight vectors as follows:

$$W_p(t+1) = W_p(t) + \alpha(t)\big[P_j - W_p(t)\big] \quad \text{for } \pi_p \in N_k \tag{2}$$

If this updation is continued for several iterations then the weights tend to approximate the distribution of input vectors in an orderly fashion. It should be noted here that the processors do not move physically in the ordering process. It is the weight vectors that are made to change to define the ordering.

One presentation each of all the points in S makes one *sweep* consisting of N iterations. After one sweep is completed, iteration for the next sweep starts again from P_1 through P_N. Several sweeps make one *phase*. One phase is completed when the weight vectors of the current set of processors have converged, that is, when

$$\| W_i(t) - W_i(t+1) \| < \epsilon \quad \text{for all } i \tag{3}$$

where ϵ is a predetermined small positive value. Only at the end of a phase, are processors inserted or deleted. That is, during the iterative process in a phase, the set of processors remains unchanged, though the weight vectors of the processors keep changing. Suppose, after the sth phase is completed, the weight vectors of the processors are $W_1(t_s), \ldots, W_{n(s)}(t_s)$ where $n(s)$ is the number of processors during the sth phase and t_s is the total number of iterations needed to reach the end of the sth phase. If now, at the end of a phase, the weight vectors of two neighbouring processors become very close, the processors are merged. If their weight vectors are far apart, a processor is inserted between them.

More formally, if

$$\| W_k(t_s) - W_{k+1}(t_s) \| = \min_{i=1,\ldots,n(s)-1} \| W_i(t_s) - W_{i+1}(t_s) \| < \delta_1 \tag{4}$$

then the two processors π_k and π_{k+1} are merged and the new processor has the weight vector as $[W_k(t_s) + W_{k+1}(t_s)]/2$. If, on the other hand,

$$\| W_l(t_s) - W_{l+1}(t_s) \| = \max_{i=1,\ldots,n(s)-1} \| W_i(t_s) - W_{i+1}(t_s) \| > \delta_2 \tag{5}$$

then one processor is inserted between π_l and π_{l+1} and the new processor has the weight vector as $[W_l(t_s) + W_{l+1}(t_s)]/2$. δ_1 and δ_2 are two predetermined positive quantities such that $\delta_1 < \delta_2$.

After the insertion and merging of processors, the next phase starts with the new set of processors. The process continues until, at the end of a phase,

$$\delta_2 \geq \| W_i(t) - W_{i+1}(t) \| \geq \delta_1 \quad \text{for all } i \tag{6}$$

The condition (6) means that the weight vectors of no two neighbouring processors are either too close or too far apart. The processors (on the basis of their weight vectors) at this stage give an approximate global shape of the input pattern. From a property of the Kohonen's feature map, this shape approximation does not depend on the starting weight vectors of the processors.

The Algorithm

Step 1: [Initialization]
Initialize $t = 0$;
Initialize the weight vectors $W_i(t)$, $i = 1, 2, \ldots, n$ to random values.

Step 2: [Sweep]
For all input patterns P_j, $j = 1, 2, \ldots, N$
Update weight vectors according to rule (2).

Step 3: [Phase]
If condition (3) is not true then goto Step 2.

Step 4: Merge or insert according to condition (4) or (5).
Step 5: If condition (6) is not true goto Step 2.
Step 6: Stop.

2.2. Tree-like patterns

In this subsection we deal with patterns that have branchings, forks or crossings (now onwards we shall call these three terms by a single term *junctures*). For example, character patterns '*Y*', '*T*', '*X*' have such properties. It can be seen that a linear structure for the processors as mentioned earlier will not work for such patterns because a processor representing a juncture in the input pattern should have more than two neighbours. Thus we need to have a nonlinear net structure. That is, in the net, a processor can have more than two neighbours. In the case of arc patterns the number of neighbours (call it *degree* of the processor) for each processor was known and was fixed during the learning process. But now a processor (representing a juncture in the pattern) can have a variable number of neighbours and the number is unknown. So the major issue in this subsection is how we can adapt the degree of each processor while learning the shape of the pattern.

Let us consider a typical tree-like pattern, e.g. the pattern '*T*' or '*Y*'. As initially we do not have any topological information of the pattern, we start with a linear net. The pattern has one juncture from which there are three branchings. After a number of iterations the net will be arranged in such a way that the two ends of the net will tend to two of the three ends of the pattern and some of the non-end processors will tend to the third end. Thus a portion of the net will form a spike shape (Figs. 1(a), (c)). Hence some processor will form a significantly small acute angle with its two neighbours. This happens because by properties of Kohonen's feature map, as mentioned earlier, the net

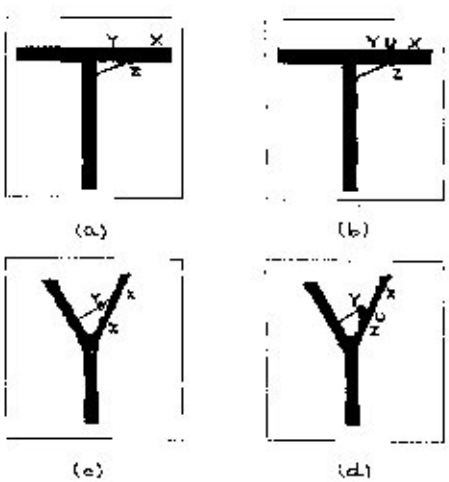


Fig. 1. Two patterns '*T*' and '*Y*'. (a) and (c): the net forms a spike to indicate one of the branchings, (b) and (d): a processor *U* with higher degree is inserted to accommodate the branching.

will span the entire range of input pattern space and will try to be within the pattern. At the same time it will preserve the topology of the pattern. Suppose a processor X forms a significantly small acute angle (decided on the basis of some threshold) with neighbouring processors Y and Z. This suggests that, in the pattern, there is a juncture from which there is another branching towards X. Hence we take the following actions to accommodate the branching when the processor X forms a small acute angle with its neighbours Y and Z (Figs. 1(b), (d)):

(a) Create a new processor U in the middle of Y and Z.

(b) Delete the link between X, Y and the link between X, Z.

(c) Establish links between U, X, between U, Y and between U, Z.

Thus we create a new processor U with a higher degree which corresponds to a juncture in the pattern. The same actions are taken for all the processors forming a significantly small acute angle. These actions are taken after a phase is complete, to adapt the topology and then the subsequent phases are continued to enable the net to approach towards a closer approximation of the shape of the pattern. For insertion and deletion of processors and for convergence of the algorithm, similar principles used in the case of arc patterns are followed here. For this the conditions 4, 5 and 6 are modified to accommodate all possible pairs of neighbouring processors as follows:

$$\text{If } \| W_k(t_s) - W_{k'}(t_s) \| = \min_{\substack{i=1,\ldots,m(s)-1 \\ \pi_\ell \in N_i - \{\pi_i\}}} \| W_i(t_s) - W_\ell(t_s) \| < \delta_1 \tag{7}$$

then the two processors π_k and $\pi_{k'}$ are merged and the new processor has the weight vector as $[W_k(t_s) + W_{k'}(t_s)]/2$. If, on the other hand,

$$\| W_i(t_s) - W_\ell(t_s) \| = \max_{\substack{i=1,\ldots,m(s)-1 \\ \pi_\ell \in N_i - \{\pi_i\}}} \| W_i(t_s) - W_\ell(t_s) \| > \delta_2 \tag{8}$$

then one processor is inserted between π_i and π_ℓ and the new processor has the weight vector as $[W_i(t_s) + W_\ell(t_s)]/2$.

The next phase starts with the new set of processors after insertion and merging. The process continues until, at the end of a phase,

$$\delta_2 \geq \| W_i(t) - W_\ell(t) \| \geq \delta_1 \text{ for all } i \text{ and } \pi_\ell \in N_i - \{\pi_i\} \tag{9}$$

2.3. Patterns with loops

The techniques discussed so far will not work for patterns that contain loops (e.g., character patterns 'A', 'B', 'D', 'P'). Let us look at pattern 'A'. Our algorithm can generate, on the basis of the principles discussed in Section 2.2, an incomplete skeleton as shown in Fig. 3(d). We are yet to complete the loop by means of bridging the gap (between processors E and F in Fig. 3(d)). This section proposes a method of doing this. In fact, the asymptotic values of the weight vectors constitute some kind of vector quantization [6]. In particular, the distance (Euclidean) measure and the updation rules as considered in our algorithm, induce a partition $\{S_i\}$ of the input pattern space specified as

$$S_i = \{P_j \in S \mid \text{dist}(P_j, W_i) \leq \text{dist}(P_j, W_k) \text{ for all } k\}$$

The above partition can be looked upon as a Voronoi tessellation which, in the present situation, means partition of the input pattern space into regions within each of which all input vectors have the same weight vector as their nearest one. Therefore each S_i is associated with a single processor. Hence the input pattern vectors can be easily labelled according to the S_i to which it belongs. In other words, each input vector is given a label according to its nearest processor. Such input labelling has also been used by Sabourin and Mitiche [13].

Definition. When the input pattern is a binary image, two processors π_i and $\pi_j (i \neq j)$ are said to be *adjacent* if there exists at least one pair of object pixels $P \in S_i$ and $Q \in S_j$ such that P and Q are 8-neighbours of each other.

Thus after convergence (let us call it initial convergence) as mentioned in Section 2.1 we label the input vectors as mentioned above and then check for each end processor

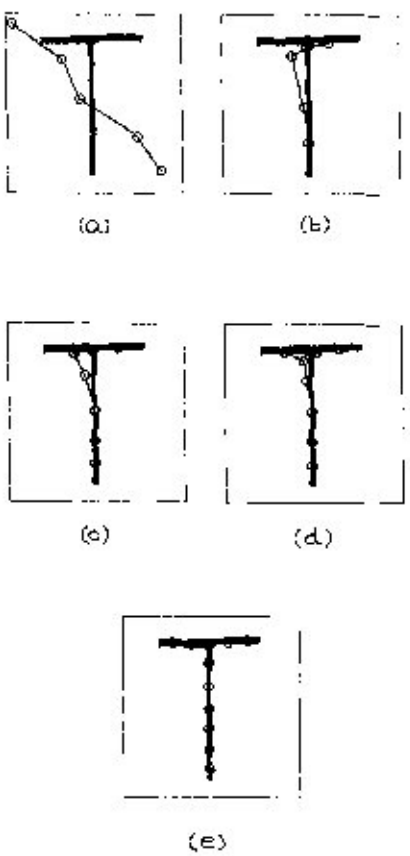


Fig. 2. Different stages of the net for the pattern '*T*'. (a) initial net; (b) after 28 sweeps; (c) after 57 sweeps, (d) after 64 sweeps; and (e) after 105 sweeps.

(having exactly one neighbouring processor) whether it is adjacent to any processor other than its neighbour. If yes, then introduce a link between these two processors. In Fig. 3(d), processor E is adjacent to processor F and they are not neighbours to each other. So, they are made connected and become neighbours (Fig. 3(e)). After this we continue the algorithm until the final convergence is reached.

3. Results and conclusions

The present paper discusses how to change adaptively the neighbourhoods of Kohonen's self-organizing neural networks model to get an appropriate model for shape extraction of binary images. To achieve the adaptibility of the neighbourhoods we start with a net having the simplest possible structure and a small number of processors. Such a structure can deal with arc patterns only. Next we explain how to deal with more complicated tree-like patterns. Subsequently patterns that contain loop structures are considered. In all these cases, in the initial net, each processor (except the two end ones)

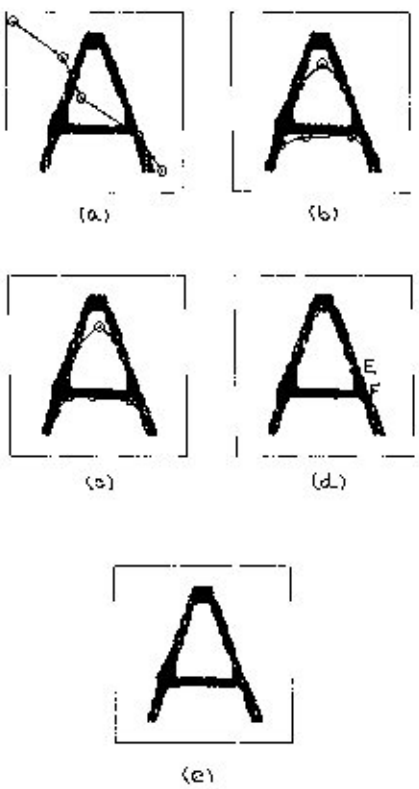


Fig. 3. Different stages of the net for the pattern 'A'. (a) initial net (b) after 57 sweeps; (c) after 68 sweeps (d) after 133 sweeps; and (e) after 191 sweeps.

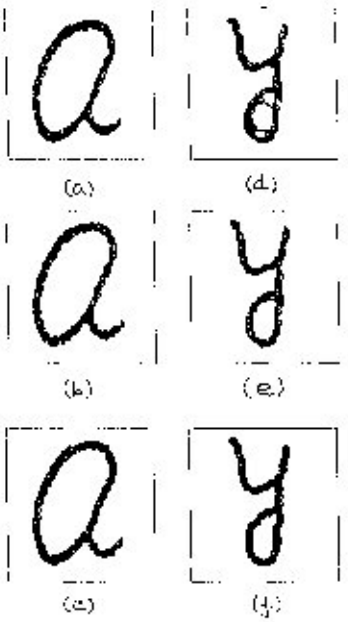


Fig. 4. Different stages of the net for cursive patterns 'a' and 'y'. (a) after 124 sweeps; (b) after 182 sweeps; (c) after 212 sweeps; (d) after 102 sweeps; (e) after 162 sweeps; (f) after 219 sweeps;

has only two neighbours. During the self-organizing process some of the processors acquires more than two neighbours, if necessary, and thus adapt their neighbourhoods dynamically on the basis of the input. Thus the net not only grows in size but also the neighbourhoods of the processors can change according to the local topology of the input pattern.

The algorithm described in the paper has been tested on a number of character patterns and some of them are presented in Figs. 2–5. Figs. 2–4 illustrate the convergence of the net to the final shape giving intermediate configurations of the net. Fig. 5 shows only the initial and final nets. In the figures, a circle represents a processor and a line joins two circles if the corresponding processors are neighbours to each other. In all the input patterns the initial net is taken to be the same. The value of initial α was 0.01 and $\alpha(t) = 0.01/[1 + t/1000]$.

It is to be noted that the parameters δ_1 and δ_2 control the average distance between two neighbouring processors. Higher values of δ_1 and δ_2 mean less number of processors and lower values of δ_1 and δ_2 mean more number of processors. Experimental results have shown that, as it can be conjectured, small δ_1 and δ_2 cause a zigzag skeleton (Figs. 6(a), (b)) and large δ_1 and δ_2 produce a skeleton that may not totally lie within the object (Fig. 6(f)). It can be seen that $\delta_1 = 5$, $\delta_2 = 10$ (Fig. 6(c)) or $\delta_1 = 6$, $\delta_2 = 12$ (Fig. 6(d)) is the right choice. Other choices of δ_1 and δ_2 are either too small or too high for the given pattern. The proper values of δ_1 and δ_2 depend on the thickness and curvature of the pattern. The parameters δ_1 and δ_2, in the present work, are chosen manually and their optimum values are judged subjectively. Further work to

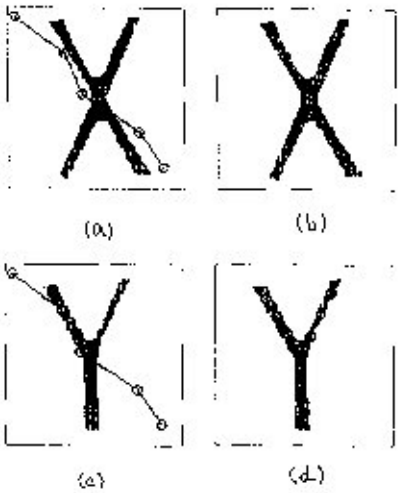


Fig. 5. Initial and final stages of the net for the patterns 'X' and 'Y'. (a) initial net; (b) after 145 sweeps for 'X'; (c) initial net; (d) after 113 sweeps for 'Y'.

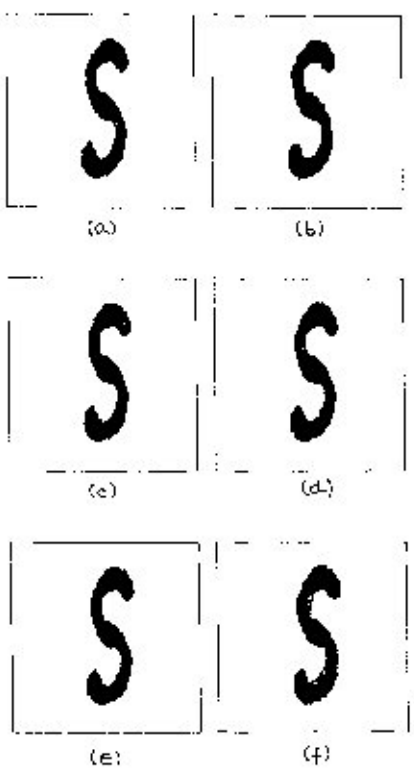


Fig. 6. For different choices of δ_1 and δ_2 the net gives different shapes. (a) $\delta_1 = 3$, $\delta_2 = 6$; (b) $\delta_1 = 4$, $\delta_2 = 8$; (c) $\delta_1 = 5$, $\delta_2 = 10$; (d) $\delta_1 = 6$, $\delta_2 = 12$; (e) $\delta_1 = 7$, $\delta_2 = 14$; (f) $\delta_1 = 8$, $\delta_2 = 16$.

automatically adapt the optimum or near optimum values of these parameters is an issue of future research.

The skeletonization technique discussed here can be extended for implementation for gray level images. The extension can be done in the following way. Suppose for a gray level pattern, g_{rs} is the gray value of the pixel at the rth row and sth column. Then the updation rule replacing Eq. (2) will be

$$W_p(t+1) = W_p(t) + \alpha(t)\left[P_j - W_p(t)\right] g_{rs}, \quad \text{for } \pi_p \in N_k$$

Multiplication by g_{rs} means that the amount of updation will be more for pixels with high gray level values and less for pixels with low gray level values. The gray level extensions for arc and tree-like patterns seem to be straightforward. But for loop patterns the extension may not be trivial because the definition of adjacency is not well defined in that case.

There are many non-neural algorithms to find skeletons of binary images. However, neural approach to skeletonization has many advantages over non-neural methods. Skeletonization being a time consuming process requires massive parallelism for faster computation which can be achieved in neural networks. In non-neural method, 2D array image processors can be used for fast processing. But a large number of processors in that case may remain idle during processing. In non-neural methods, output skeletons are in many cases rotation-dependent while in the present neural method the output skeleton is rotation-independent. The non-neural methods (for example, template matching methods) are usually sensitive to presence of boundary noise in the input pattern while our neural method, as can be seen, is robust to such noise. The non-neural methods are not generally extendable to gray level images while the present neural method is extendable.

Acknowledgments

The authors thank the reviewers for their comments which led to considerable improvements of the manuscript.

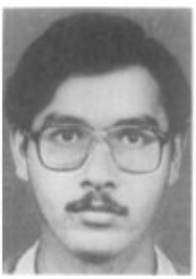
References

[1] D.H. Ballard and C.M. Brown, *Computer Vision* (Prentice-Hall, NJ, 1982).
[2] G.A. Carpenter and S. Grossberg, eds., *Neural Networks for Vision and Image Processing* (MIT Press, 1992).
[3] D. Choi and S. Park, Self-creating and organizing neural networks, *IEEE Neural Networks* 5 (4) (1994) 561–575.
[4] B. Fritzke, Let it grow – self-organizing feature maps with problem dependent cell structure, in: T. Kohonen et al., eds., *Artificial Neural Networks*, Vol. 1 (North-Holland, 1991) 403–408.
[5] J.A. Kangas, T. Kohonen and J. Laaksonen, Variants of self-organizing maps, *IEEE Neural Networks* 1(1) (1990) 93–99.
[6] T. Kohonen, *Self-Organization and Associative Memory* (Springer-Verlag, 1989).
[7] R.P. Lippmann, An introduction to computing with neural nets, *IEEE ASSP Mag.* (Apr. 1987) 4–22.
[8] Y.H. Pao, *Adaptive Pattern Recognition and Neural Networks* (Addison-Wesley, 1989).

[9] S.K. Parui, A. Datta and T. Pal, Shape approximation of arc patterns by dynamic neural networks, *Signal Processing* 42(2) (1995) 221–225.
[10] T. Pavlidis, *Algorithms for Graphics and Image Processing* (Springer-Verlag, Berlin, 1982).
[11] A. Rosenfield and A.C. Kak, *Digital Picture Processing* (Academic Press, UK, 1982).
[12] D.E. Rumelhart and J.L. McClelland, eds., *Parallel Distributed Processing*, Vol. 1 (MIT Press, Cambridge, 1986).
[13] M. Sabourin and A. Mitiche, Modeling and classification of shape using a Kohonen associative memory with selective multiresolution, *Neural Networks* 6 (1993) 275–283.

Amitava Datta received his Master's degree from the Indian Statistical Institute, Calcutta in 1977 and did a post-graduate course in Computer Science from the same institute in 1978. After working for a few years in reputed computer industries he joined the Computer Centre of the Indian Statistical Institute as a System Analyst in 1988. Since then he has been working in image processing and pattern recognition. From 1991 to 1992 he visited GSF, Munich as a Guest Scientist and worked on query-based decision support system. His current interests are in neural net based image processing and computer vision.

S. K. Parui received his Master's degree in Statistics from the Indian Statistical Institute, Calcutta in 1975. After a brief career in commercial software development, he was back to the academics and received his Ph.D. degree in image processing from the Indian Statistical Institute in 1986. From 1985 to 1987 he held a post-doctoral position in Leicester Polytechnic, England to work on an automatic industrial inspection project. In 1987 he joined the Indian Statistical Institute as a Lecturer. During 1988–1989 he was a Visiting Scientist in GSF, Munich working on a pattern recognition problem in biomedicine. He is now an Associate Professor in the Indian Statistical Institute. His current research interests include shape analysis, image processing, pattern recognition and neural networks.



Tamaltaru Pal received his B.Sc. degree in Statistics in 1989 and Master's degree in Computer Applications in 1992 from the Calcutta University. He is now in the Computer Vision & Pattern Recognition Unit of the Indian Statistical Unit. His research interests include pattern recognition, neural networks and image processing.