A Novel SOM-based Approach for Active Contour Modeling

Y. V. Venkatesh, S. Kumar Raja and N. Ramya
Computer Vision Laboratory
Department of Electrical Engineering,
Indian Institute of Science,
Bangalore-560012.

Abstract

In this paper, we integrate the advantages of SOM- and snake-based ACMs in order to extract the desired contour from images. We employ (i) the feature points to guide the contour, as in the case of SOM-based ACMs; and (ii) the gradient and intensity variations in a local region to control the contour movement. However, in contrast with the snake-based ACMs*, we do not use an explicit energy functional based on gradient or intensity. The algorithm is tested on synthetic binary and gray-level images, and the results show the superiority of the proposed algorithm over other conventional SOM- and snake-based ACM algorithms.*

1. INTRODUCTION

In many tasks related to computer vision, edge detection and the associated operation of edge linking constitute the preliminary step. Boundaries detected by such techniques often do not conform to the true boundaries of objects in images, as they are sensitive to (image) noise and intensity variations. In order to overcome this deficiency, Kass *et al.* proposed a flexible framework, called snake, with scope for higher level image interpretation. Such models, in which the contours deform themselves in the process of matching, are called Active Contour Models (ACM).

The *snake* [1] is probably the first proposed ACM, which is a controlled continuity-spline under the influence of internal (spline), image and external (constraint) forces. The problem of contour modeling is then cast in the framework of energy minimization, with the energy functional consisting of terms corresponding to the internal, image and external forces. Since then, many modifications have been suggested to this model. (See [2], [3]) *The main drawback of such* snake *models [1] is that they get stuck in a local minima, and, as a result, cannot move into the boundary concavities. Furthermore, they require the initial contour to be close to the true boundary for proper convergence of the snake*.

An alternative approach to the (above-described parametric) active contour models is a *neural network* based on a self-organizing map (SOM) [4], [5]. The network consists of two layers: an input layer (to receive the input data), and an output layer. The input data are usually the coordinates of feature points. The network is trained in an unsupervised manner in such a way that its output moves towards the nearest salient contour.

In [4], the SOM-based network with a fixed number of neurons in a chain topology is used to model the contour. This model, in common with most of the ACMs, requires an initial contour, starting from which it evolves. A neural network isomorphic to this initial contour is constructed, and subjected to deformation in order to map onto the nearest salient contour in the image. The SOM is trained using a scheme similar to Kohonen's algorithm, but is different from the classical SOM in terms of its architecture. At the end of the training, the output of the network converges to the actual contour. It is to be noted that the SOM-based algorithm requires that the initial contour should be *not only* close to the object boundary but also similar to the shape of the object.

The *time-adaptive self-organizing map* (TASOM) [5] is a modified form of the SOM-algorithm that overcomes the above limitations. Unlike the SOM, which has a time-decreasing learning rate and neighborhood parameters common to all neurons, the TASOM algorithm employs individual learning rates and neighborhood parameters for each neuron. These parameters are updated on the basis of the environment conditions in each iteration of the TASOM learning algorithm. Moreover, the TASOM incorporates a feature of insertion and deletion of neurons which ensures the continuity of the contour.

However, in [4] and [5], the movement of the contour is solely based on the edge map (feature points), as a result of which the contour leaks through the boundary in the case of broken or weak edges. Also, as all the feature points are allowed to compete for any neuron, the TASOM algorithm gives unsatisfactory results in case the edges are thick or have very high concavities.

We propose an algorithm that not only combines the merits of both SOM- and snake-based ACMs but also overcomes the limitations of TASOM. The rest of the paper is organized as follows: after motivating and describing the algorithm in the next section (Sec. 2), we present the results of computer implementation in Sec. 3. Finally, we conclude the paper in Sec. 4.

0-7803-8894-1/04/$20.00 © 2004 IEEE

ISSNIP 2004

2. PROPOSED ALGORITHM

The proposed algorithm is motivated by the following observations. The movement of the contour in SOM-based ACM is solely guided by feature points extracted from the image such as edge maps. Hence, if some of the feature points are faulty, as in the case of a broken edge map, the contour may not converge to the desired boundary. *In contrast*, the snake-based ACM uses explicitly the gradient information for the movement of the contour, and as a consequence, such faults do not occur. However, the limitations of the snake-based ACM are: (i) it requires the initial contour to be very close to the desired boundary; and (ii) it cannot invade boundary concavities, and may get stuck in a local minimum of an energy functional.

The **proposed algorithm** uses a neural architecture similar to that of the SOM-based ACM, but *in addition to the feature points, it uses (i)* **intensity variations***, and (ii)* **gradient information** *in a local region in order to guide the movement of the contour*. The two essential ideas used for controlling the neuronal weight updates are as follows:

1) If a neuron is near a region boundary, it is forced to move along the normal to the contour or opposite to it, such that the gradient magnitude at the new location is greater than that at the current location. As in the case of snakes, where gradient energy along the contour is calculated to move the contour to the desired boundary (i.e., to a location where the energy functional is smaller), *the neuron is not allowed to move to the new position if there is a decrease in the gradient magnitude*. Such a movement of the contour ensures proper convergence even if there are broken or weak[1] edges. Note that **this approach does not use an explicit gradient energy term as in the case of snake-based ACM**.
2) On the other hand, if the neuron is in a uniform region, it is moved closer to an appropriately chosen feature point.

Further, in the proposed algorithm, the *updation procedure of neuron weights is* unlike *Kohonen's, and is specific only to the problem of active contours*. The TASOM algorithm [5] uses a set of (i) control points which give the position of the contour and (ii) weights which give the direction in which the contour should move. The amount of movement is controlled by the speed parameter ρ. The control points are copied to the weight after every epoch (i.e., after all feature points are presented to the network) for further updation until convergence. In contrast with TASOM, the proposed algorithm uses the **weights** of the neurons themselves as the **new positions** of the control points, and *hence* of the contour.

A. Description

The output of an edge detection algorithm (as applied to the image) provides the feature points for training the neural network. Let $\mathscr{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m\}$ be the set of the feature points, where $\mathbf{x}_k = (x_k, y_k)$ are the $x-$ and $y-$coordinates of the k^{th} feature point.

At epoch n, the contour to be deformed is modeled as a *sequence* of control points $\mathcal{P}(n) = \mathbf{p}_1, \mathbf{p}_2, \ldots, , \mathbf{p}_{M(n)}$, where $\mathbf{p}_j = (x_j, y_j)$ is the position of the j^{th} control point, and $M(n)$ is the number of control points in the n^{th} epoch. In this algorithm, a set of neurons arranged in chain topology similar to that found in [4], [5] is used. Let $\mathcal{W}(n) = \{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_{M(n)}\}$ be the sequence of neuronal weights, where $\mathbf{w}_i = (w_i^x, w_i^y)$ is the weight of i^{th} neuron. *Note that the number of nodes in the network is equal to the number of the control points, i.e., there is one-to-one correspondence between each of the control points and neurons*[2] *in the network* (isomorphism).

The proposed algorithm involves, in brief, the following steps:

1) Initialization of the weights of the network to the initial position of the contour: $\mathcal{W}(0) \leftarrow \mathcal{P}(0)$.
2) Initialization of the updates of the neuronal weights to zero.
3) Finding the winning neuron for every feature point.
4) Computation of updates for winner neurons and validation of the updates.
5) Parallel updation of the weights of the network, and updation of the control points: $P(n) \leftarrow \mathcal{W}(n)$.
6) Insertion and deletion of neurons to the network.
7) Updation of the learning rate parameter η.
8) Repetition of the algorithm from Step 2 until the convergence criterion is satisfied.

In Step 1, a neural network *isomorphic* to the initial contour is constructed, and the weights of the neurons are initialized to the initial position of the contour. Subsequently, the updates for the neuronal weights are set to 0 (Step 2). Description of Step 3 needs the following definitions of certain functions and sets:

For every feature point, there exists a neuron which is physically nearest (in Euclidean distance) to it, and we term these winning neurons as *first-level winners* (FLW). The winner neuron associated with the feature vector $\mathbf{x}_k$ is denoted as

$$n(\mathbf{x}_k) = \arg\min_j d(\mathbf{x}_k, \mathbf{w}_j), \tag{1}$$

where $d(\mathbf{x}, \mathbf{w})$ is the Euclidean norm between the feature vector and the neuron weight. It is possible that many feature points find the same neuron as a first-level winner. For every FLW i, we associate a set of feature points $\mathbb{X}(i)$ which makes the neuron a winner according to (1). In other words,

$$\mathbb{X}(i) = \{\mathbf{x} \,|\, i = n(\mathbf{x}), \mathbf{x} \in \mathscr{X}\}. \tag{2}$$

It is clear that $\mathbb{X}(i) \subset \mathscr{X}$.

Also, for each neuron i, there exists a feature point $\mathbf{g}(i)$ among all feature points $\mathscr{X}$ which is nearest to it, i.e.,

$$\mathbf{g}(i) = \arg\min_{\mathbf{x} \in \mathscr{X}} d(\mathbf{x}, \mathbf{w}_i). \tag{3}$$

[1] Weak edges are those which are perceived by the human eye as true edges but have small gradient magnitude values.

[2] We use the words "node" and "neuron" interchangeably.

Note that $\mathbf{g}(i)$ *may not belong to* $\mathbb{X}(i)$. For each FLW i, we can find a feature point from the set $\mathbb{X}(i)$ which is physically nearest to it. We denote this feature point as $\mathbf{f}(i)$, as given by

$$\mathbf{f}(i) = \arg \min_{\mathbf{x} \in \mathbb{X}(i)} d(\mathbf{x}, \mathbf{w}_i). \tag{4}$$

It is obvious that $d(\mathbf{f}(i), \mathbf{w}_i) \geq d(\mathbf{g}(i), \mathbf{w}_i)$.

After finding the FLWs for all the feature points in Step 3, we compute their (i.e., FLWs') updates in Step 4 as follows:

$$\Delta \mathbf{w}_i(t) = \eta(t)\, p\, (\mathbf{f}(i) - \mathbf{w}_i(t)), \tag{5}$$

where $\eta(t)$ is the learning rate parameter, and p is the damping factor given by

$$p = \frac{\|\mathbf{g}(i) - \mathbf{w}_i(t)\|}{\|\mathbf{f}(i) - \mathbf{w}_i(t)\|}. \tag{6}$$

The updation equation (5) ensures that the first-level neuron i moves not only towards that feature point which finds it as a winner, but also is closest to itself according to (4). *This rule of updation is distinct from other conventional SOM-based ACM implementations.* Note that $0 < p \leq 1$. The role of the damping factor is to ensure a smooth evolution of the contour towards the object boundary, without penetrating it. Before updating, we validate the (*currently*) computed update for every neuron. This step is essential in order to prevent the contour from penetrating weak or broken edges. The unit vector $\mathbf{e}$ tangential to the contour at the position of the neuron i is defined as

$$\mathbf{e} = (e_x, e_y) = \frac{\mathbf{w}_{i+1} - \mathbf{w}_{i-1}}{\|\mathbf{w}_{i+1} - \mathbf{w}_{i-1}\|}.$$

The unit vector perpendicular to the above vector (i.e., normal to the contour) is given by

$$\mathbf{e}^{\perp} = (-e_y, e_x).$$

For $k \in \{1, 2, \ldots N_r\}$, consider the pair of points

$$\begin{aligned} \mathbf{p}_a &= \mathbf{w}_i + k\, \mathbf{e}^{\perp}, \\ \mathbf{p}_b &= \mathbf{w}_i - k\, \mathbf{e}^{\perp}, \end{aligned} \tag{7}$$

where N_r is the number of neighborhood points considered for determining a region boundary. Let "sgn" denote the signum function, and I, the image intensity function. We define the quantity

$$u(i) \triangleq \sum_{k=1}^{N_r} \mathrm{sgn}\left(I(\mathbf{p}_a) - I(\mathbf{p}_b)\right). \tag{8}$$

If a neuron is near a region boundary, then the sign of the difference between the image intensities at the points $\mathbf{p}_a$ and $\mathbf{p}_b$ must be the same for all k. Therefore, for a neuron which is near a region boundary, u defined in (8) must be N_r or $-N_r$. In our work, we have chosen $N_r = 2$. Note that if $N_r = 1$, then $u(i)$ will always be trivially equal to 1 (see (8)). Hence N_r should be ≥ 2. Larger values of N_r can be used for noisy images. Moreover, in order to make the algorithm robust with respect to noise or minor intensity variations, we consider a neuron to be in a uniform region if $|I(\mathbf{p}_a) - I(\mathbf{p}_b)| < G_{\max}$ for any k. In our experiments, we have explored values of $G_{\max}$ between 1 and 5.

In order to declare whether a neuron is in a homogenous or uniform region, we define the function $R(i)$ as follows:

$$R(i) \triangleq \begin{cases} 1 & : \quad |u(i)| = N_r \\ 0 & : \quad |I(\mathbf{p}_a) - I(\mathbf{p}_b)| < G_{\max} \text{ for any } k \\ 0 & : \quad \text{otherwise.} \end{cases} \tag{9}$$

If $R(i) = 0$, then the neuron is considered to be in an uniform region, and its update is set according to (5). However, near a region boundary (i.e., for $R(i) = 1$), we consider two cases;

1) $\|\Delta \mathbf{w}_i(t)\| > 1$; and
2) $\|\Delta \mathbf{w}_i(t)\| \leq 1$.

We define the quantities

$$\begin{aligned} \mathbf{u}_i &= \frac{\Delta \mathbf{w}_i(t)}{\|\Delta \mathbf{w}_i(t)\|}, \\ h &= \left|\mathbf{e}^{\perp} \,.\, \mathbf{u}_i\right|. \end{aligned}$$

Case 1: ($\|\Delta \mathbf{w}_i(t)\| > 1$)

In this case the neuron is forced to move along the direction $\mathbf{e}^{\perp}$ (i.e., normal to the contour) or opposite to it ($-\mathbf{e}^{\perp}$). For a neuron i, consider the quantities,

$$\begin{aligned} q_{-1} &= \|\nabla I(\mathbf{w}_i - h\, \mathbf{e}^{\perp})\|, \\ q_0 &= \|\nabla I(\mathbf{w}_i)\|, \\ q_1 &= \|\nabla I(\mathbf{w}_i + h\, \mathbf{e}^{\perp})\|, \\ q_u &= \|\nabla I(\mathbf{w}_i + h\, \mathbf{u}_i)\|, \end{aligned} \tag{10}$$

where $\|\nabla I(.)\|$ denotes the gradient magnitude function; q_0 is the gradient magnitude at the location of the neuron; q_{-1} & q_1 are the gradient magnitudes at points left and right of the neuron (by convention), along the contour normal; and q_u is the gradient magnitude at a point along the update direction.

Our objective is to make all neurons move towards gradient peaks, and thereby towards the true boundary. To this end, we do not allow any neuron to move to a point where the gradient magnitude is smaller than at its current location. The weight update for a FLW (i.e., first-level winner, as defined earlier) neuron is computed as follows:

If $(q_1 < q_0)$ & $(q_{-1} < q_0)$
  $\Delta \mathbf{w}_i(t) = 0$
  else if $(q_{-1} \leq q_0 < q_1)$
    $\Delta \mathbf{w}_i(t) = h\, \mathbf{e}^{\perp}$
    else if $(q_1 \leq q_0 < q_{-1})$
      $\Delta \mathbf{w}_i(t) = -h\, \mathbf{e}^{\perp}$
      else if $(q_0 > q_u)$
        $\Delta \mathbf{w}_i(t) = 0$
        else $\Delta \mathbf{w}_i(t) = h\, \mathbf{u}_i$.

The first condition does not allow the neuron to move from a local maximum of the gradient magnitude. The second, third

and fifth conditions ensure that the neuron moves a towards a gradient peak. The fourth condition prevents the neuron from moving from a higher gradient location to a lower one.

Case 2: ($\|\Delta \mathbf{w}_i(t)\| \leq 1$)

Here the neuron is moved along the update direction provided there is an increase in the gradient magnitude. For a neuron i, consider the quantities,

$$\begin{array}{rcl} q_{-1} & = & \|\nabla I(\mathbf{w}_i - \mathbf{e}^{\perp})\|, \\ q_0 & = & \|\nabla I(\mathbf{w}_i)\|, \\ q_1 & = & \|\nabla I(\mathbf{w}_i + \mathbf{e}^{\perp})\|, \\ q_u & = & \|\nabla I(\mathbf{w}_i + \Delta \mathbf{w}_i(t))\|. \end{array} \tag{11}$$

The weight update for a FLW neuron is computed as follows:

If $(q_0 > q_u)$
  $\Delta \mathbf{w}_i(t) = 0$
  else if $(q_1 < q_0)$ & $(q_{-1} < q_0)$
    $\Delta \mathbf{w}_i(t) = 0$
      else $\Delta \mathbf{w}_i(t) = \eta(t)\, p\,(\mathbf{f}(i) - \mathbf{w}_i(t))$

Similar to Case 1, the first and second conditions do not allow a neuron to move either to a position of lower gradient magnitude or away from a gradient peak.

The maximum update among all the first-level winner updates (according to (5) and validation) is found, and if the maximum update is very small, then the contour does not move significantly. In this case, the second-level winners (SLW) are invoked.

The SLWs are picked from a set of neurons which have not won in the competition in the previous iteration, i.e., these neurons are not FLWs, and *they are not located near region boundaries* ($R(i) = 0$). The feature points which are used to find this set of winners are those whose distances from their corresponding FLW neurons are greater than $\sqrt{2}\, d_{\max}$. *This constraint is used for preventing feature points which are very close to FLW neurons to have SLWs*. Let the feature $\mathbf{x}_k$ satisfy the property

$$\min_{j \in \mathscr{X}} d(\mathbf{x}_k, \mathbf{w}_j) > \sqrt{2}\, d_{\max}.$$

The set of neurons *which are not FLWs*, and for which $R(i) = 0$ is denoted by $\mathscr{N}_s$. Mathematically, a neuron $s \in \mathscr{N}_s$ is a SLW of feature $\mathbf{x}_k$ if

$$d(\mathbf{x}_k, \mathbf{w}_s) \leq d(\mathbf{x}_k, \mathbf{w}_j), \quad \forall j \in \mathscr{N}_s. \tag{12}$$

These SLW neurons have an important role to play when dealing with contours with high concavities. The updates for the SLW neurons are computed and validated in the same way as for the FLW neurons (according to (5)). All the neuronal weights are updated *in parallel* (Step 5) according to following equation:

$$\mathbf{w}_i(t+1) = \mathbf{w}_i(t) + \Delta \mathbf{w}_i(t).$$

Note that the parallel updation in our proposed algorithm is distinct from sequential updation used in the previous SOM-based ACMs [4], [5].

In Step 6, those neurons which are not modified or updated during the epoch are deleted. Since only the winning neurons (at the first and second levels) are updated, all the non-winners are deleted. Furthermore, if the distance between two *topologically neighboring* neurons is less than $d_{\min}$, then one of the two neurons is deleted. However, during deletion, if one of the neurons is FLW and the other SLW, then the latter is deleted. In case of both being FLWs or SLWs, any one of them can be deleted.

After deletion of nodes, the network contains neurons which are either FLW or SLW. If the distance between any two topologically neighboring neurons is greater than $d_{\max}$, then a "dense" set of nodes is inserted between the two nodes. Let i and j denote two topologically neighboring neurons, and $d_{ij} = \|\mathbf{w}_i - \mathbf{w}_j\|$, the distance between the two nodes. The quantity N is defined as

$$N = \left\lfloor \frac{2\, d_{ij}}{d_{\max}} \right\rfloor - 1,$$

where $\lfloor\rfloor$ is the floor operation. If d_{ij} is greater than $d_{\max}$ and $N > 0$, then the number of new neurons to be inserted between winners i and j is N. The weight of the k^{th} neuron inserted is given by

$$\mathbf{w}_{\text{new}}^{(k)} = \mathbf{w}_i + \frac{k\,(\mathbf{w}_j - \mathbf{w}_i)}{N+1}, \quad k = 1, 2, \ldots, N. \tag{13}$$

For consistency $d_{\max} > 2d_{\min}$. In order to have an inter-nodal distance which is not less than 1 pixel in the final contour, we choose the values of $d_{\min}$ and $d_{\max}$ to be 1.0 and 2.3.

In Step 7, the learning rate parameter is updated in such a way that the value of η falls very slowly until a certain number of epochs (μ), and then falls rapidly towards the final value of η. The rate at which the value of η falls is controlled by the slope parameter λ. The updation for the learning rate parameter is done as follows:

$$\eta(t) = \eta_f + (\eta_i - \eta_f)\left(\frac{1 + \exp(-\mu/\lambda)}{1 + \exp((t-\mu)/\lambda)}\right) \tag{14}$$

where η_i and η_f are the initial and final values of the learning rate parameter. Note that $0 < \eta_i,\ \eta_f\ < 1$, and $\eta_i \gg \eta_f$. In our experiments, we have chosen $\eta_i = 0.1$ and $\eta_f = 0.001$.

In Step 8, the network is checked for convergence, and if the convergence condition is not satisfied, the algorithm is repeated from Step 2. In our algorithm, we use a convergence criterion similar to that of snake-based ACM, wherein we use the gradient information to determine whether the contour approximates the desired boundary.

Note that the snake-based ACM uses a gradient energy and internal energy (smoothness) terms to determine the desired contour (corresponding to a local minima of a energy functional).

However, the proposed algorithm does not use any explicit energy term for determining the convergence.

The SOM network is said to have converged if the following criteria are satisfied:

1) *Neurons which are FLW or SLW must be at a gradient peak or have their update magnitudes less than* ϵ (for instance, $\epsilon = 0.001$). To verify the gradient peak criterion, consider a neuron i (either FLW or SLW), and the quantities q_{-1}, q_0 and q_1 defined in (11). The neuron i is said to be at a gradient magnitude peak if $q_0 > q_{-1}$ and $q_0 > q_1$. In other words, this criterion states that the gradient magnitude at the location of the neuron must be greater than at the points left and right of the neuron, along the contour normal. The updating criterion merely states that the movement of the winner neurons must be very small.
2) *Non-winner neurons must be near a region boundary*. In order to verify this criterion, consider a neuron i (non-winner), and the points $\mathbf{p}_a$ and $\mathbf{p}_b$ (defined in (7)) which are the neighbors along the contour-normal at the neuron location. Also, for $k \in \{1, 2 \ldots N_r\}$, consider another pair of points along a direction tangential to the contour:

$$\begin{aligned} \mathbf{p}_c &= \mathbf{w}_i + k\,\mathbf{e}, \\ \mathbf{p}_d &= \mathbf{w}_i - k\,\mathbf{e}. \end{aligned} \tag{15}$$

Similar to the quantity $u(i)$ in (8), we define

$$v(i) = \sum_{k=1}^{N_r} \operatorname{sgn}\left(I(\mathbf{p}_c) - I(\mathbf{p}_d)\right). \tag{16}$$

A non-winner neuron i is said to be near the region boundary, if either $|u(i)| = N_r$ or $|v(i)| = N_r$, i.e., a non-winner neuron must satisfy the region boundary condition in either the contour-normal or contour-tangential direction.

The above convergence criteria are more effective than those of ACMs based on SOM[4] and TASOM[5], for the following reasons:

1) In [4], the algorithm is terminates after a user-specified number of iterations. However, this criterion may not yield the desired contour even after the specified number of iterations.
2) In order to overcome the above problem, the algorithm in [5] checks if (i) every neuron is very near a feature point and (ii) no new neuron has been inserted in an epoch to ensure convergence. These criteria seem to be very strict, and are not satisfied when the initial rough edge map (feature points) is broken.

Our algorithm overcomes the above drawbacks of SOM and TASOM by employing weaker criteria so as to achieve proper convergence even in the case of weak or broken edges.

3. RESULTS

The proposed algorithm has been tested on the following types of images: (1) synthetic binary (for illustrating the advantages of the proposed method); (2) gray (captured in a laboratory set up) and (3) biomedical (lung).

In order to study the effects of a broken edge map, we show, in Fig. 1, a hand image with the initial contour; in Fig. 2, the set of feature points; and in Figs. 3, and 4, the results of TASOM and the proposed algorithms, respectively.

The TASOM algorithm is solely guided by the feature points, and therefore the contour leaks through the weak edges. In contrast, in the proposed algorithm, information from gradient and intensity variations is used to prevent the contour from penetrating the true boundary.

Figure 5 shows a "star image" with the initial contour; and Fig. 6, the initial set of feature points (note the thick edges). The results of our algorithm and TASOM are shown in Figs. 7 and 8, respectively. Clearly, the contour obtained using the TASOM algorithm is not satisfactory compared to that of Fig. 7. This is because, in TASOM [5], a neuron may move towards a farther feature point even though there are other feature points closer to it. However, in our algorithm, a winner neuron moves only towards that feature point which finds it as a winner, and also is closest to itself (see (5)).

Figure 9 is a synthetic spiral image depicted along with the initial contour. The contour obtained from the proposed algorithm is shown in Fig. 10. In contrast, the output of the TASOM algorithm (Fig. 11) does not correspond to the desired boundary due to high concavities in the original contour.

For the sake of completeness, we now present further results obtained from the proposed algorithm. The lung MRI image (see Fig. 12) was used as a test example in [3] (snake-based ACM). It is found that the result (see Figs. 13) obtained from our algorithm can be compared favorably with that of [3] where the authors employ the modified GVF [2].

Fig. 1: A hand image along with the initial contour

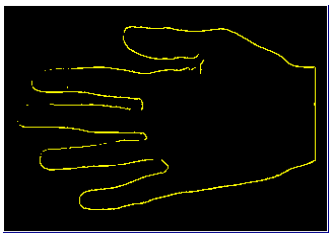

Fig. 2: The set of feature points for the hand image

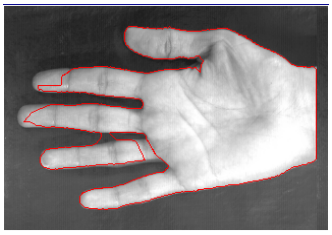

Fig. 3: Final contour obtained using TASOM

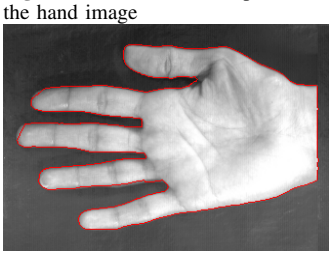

Fig. 4: Final converged contour obtained using the proposed algorithm

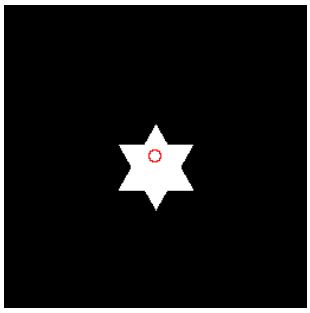

Fig. 5: "Star" image along with initial contour

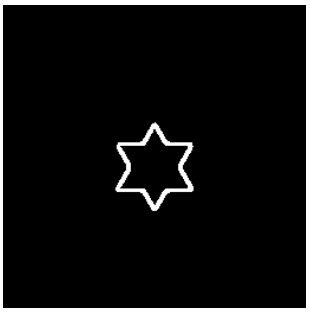

Fig. 6: Set of feature points for the star image

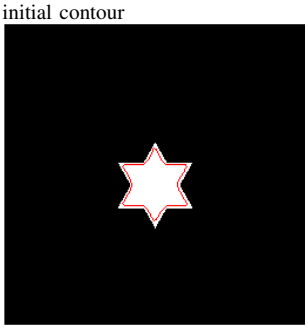

Fig. 7: Final contour obtained using the proposed algorithm

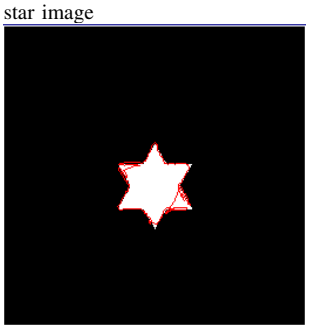

Fig. 8: Final contour obtained using TASOM

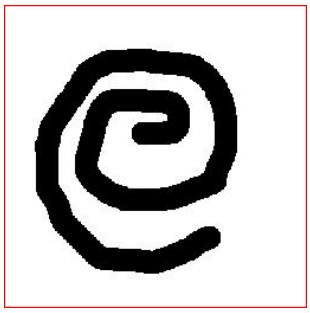

Fig. 9: "Spiral" image along with the initial contour

Fig. 10: Final converged contour obtained using the proposed algorithm

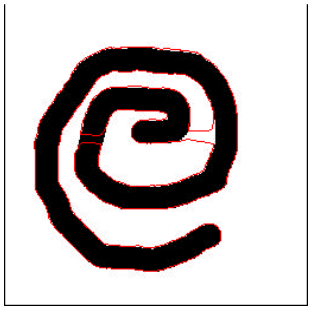

Fig. 11: Final contour obtained using TASOM

4. Conclusions

We have proposed a novel algorithm for ACM by integrating the conventional SOM and snakes. In addition to using the feature points for guiding the contour, we also employ the gradient and intensity variation information for controlling the movement of the contour. The advantages of the proposed algorithm are:

1) Every winner neuron moves only towards that feature point which finds it as a winner, and also is closest to itself.
2) In our algorithm, we have used the concept of *second-level winners* in order to facilitate the movement of the contour into boundary concavities.
3) We use (i) **intensity variations**, and (ii) **gradient information** to prevent a neuron from moving to a location of lower gradient magnitude, thereby ensuring that the contour does not penetrate the true boundary.

By applying our algorithm to (1) synthetic binary, (2) gray, and (3) biomedical images, we have demonstrated its superiority over the existing results in active contour modeling.

Fig. 12: Lung MRI image along with the initial contour

Fig. 13: Final contour obtained using the proposed algorithm

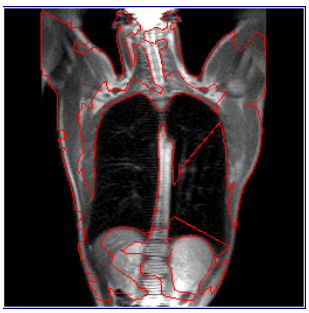

Fig. 14: Final contour obtained using TASOM algorithm

References

[1] A. Witkin M. Kass and D. Terzopoulos, "Snakes: Active contour models," *Proceedings of the First IEEE Conference on Computer Vision*, pp. 259–268, 1987.

[2] C. Xu and J. L. Prince, "Snakes, shapes and gradient vector flow," *IEEE Transactions Image Processing*, vol. 7, no. 3, pp. 359–369, March 1998.

[3] Nilanjan Ray, Scott T. Acton, Talissa Altes, Eduard E. de Lange, and James R. Brookeman, "Merging parametric active contours within homogeneous image regions for mri-based lung segmentation," *IEEE Transactions on Medical Imaging*, vol. 22, no. 2, pp. 189–199, February 2003.

[4] Y. V. Venkatesh and N. Rishikesh, "Self organizing neural networks based on spatial isomorphism for active contour modeling," *Pattern Recognition*, vol. 33, pp. 1239–1250, 2000.

[5] H. Shah-Hosseini and R. Safabakhsh, "A TASOM-based algorithm for active contour modeling," *Pattern Recognition*, vol. 24, pp. 1361–1373, 2003.