New Object-oriented Segmentation Algorithm
based on the CNN Paradigm

Giuseppe Grassi, Senior Member, IEEE, Eugenio Di Sciascio, Member, IEEE,
Luigi A. Grieco, Member, IEEE, and Pietro Vecchio

Abstract—This paper illustrates a new object-oriented segmentation algorithm based on the CNN paradigm. The approach, which exploits a rigorous model of the image contours, presents two remarkable features: i) it provides more accurate segmented objects than the ones obtained by other CNN-based techniques; ii) it makes use of CNN templates that take into account the hardware characteristics imposed by the CNNUM. Results carried out for benchmark video sequences highlight the capabilities of the proposed technique.

Index Terms—CNN, object-oriented algorithm, segmentation.

I. INTRODUCTION

It is well-known that segmentation is one of the most important image processing tasks, used in several applications including the MPEG-4 object-oriented standard [1]. Since the image-analysis operations in the object-oriented schemes require high computational power, the idea recently proposed in [2]-[6] is to exploit the Cellular Neural Network (CNN) paradigm, along with the advanced computational capabilities offered by the CNN Universal Machine (CNNUM), as the engine responsible for the image-analysis operations. In particular, in [1]-[4] CNN algorithms and templates for obtaining the segmentation of a video sequence into moving objects are introduced. This paper makes a further contribution to the topic by illustrating a new object-oriented segmentation algorithm. The proposed approach presents two remarkable features. It provides more accurate segmented objects than the ones obtained in [1]-[4], by virtue of the rigorous model of the image contours adopted in the edge extraction phase. Additionally, the proposed approach (differently from [1]-[3]) takes into account the hardware characteristics of the CNNUM. Mainly, the chip does not allow the processing of nonlinear templates and has the electrical restriction that the highest value allowed for the template coefficients is 3, while for the bias values the upper bound is 6 [4]. Moreover, the chip has a 8-bit accuracy, which poses additional constraints on template coefficients. Herein, new CNN templates that take into account these limitations have been designed. The paper is organized as follows. In Section II a model of the image contours is described, along with an edge detection algorithm based on a two concentric circular windows operator, which will be used in what follows.

In Section III the low-pass filtering is summarized, whereas Section IV illustrates a CNN algorithm for edge extraction. In particular, the proposed CNN-based technique performs a preliminary selection of edge candidates and exploits the dual window operator [7] to reveal edges as zero-crossing points of a difference function, which depends only on the minimum and maximum values in the two windows. In Section V the motion detection algorithm is described, whereas Section VI presents a CNN algorithm for object extraction. Finally, in Section VII some benchmark video sequences are analyzed, with the aim to show the advantages of the proposed technique.

II. MODEL OF THE IMAGE CONTOURS

This Section briefly revises an image processing technique (i.e., the dual window operator for edge extraction) developed by one of the authors in [7]. We start the explanation of the basic idea by considering an ideal scenario with no noise. There contours are placed exactly half way along the slopes of transition zones between almost uniform luminance areas. Edge extraction using the dual window operator is based on a criterion able to localize the mean point within the transition area between two uniform luminance areas. Let $I(x,y)$ be an input gray level image. For each sample $s \in I(x,y)$, let us consider two concentric circular windows, centered in $s$ and having radius $r$ and $R$, respectively, with $r < R$. Let us define the following quantities: $M^s$ and $m^s$, which represent the maximum and minimum values of $I$ within the window of radius $R$; $M'$ and $m'$, which represent the maximum and minimum values of $I$ within the window of radius $r$. For each sample $s$, let us define the quantities [7]:

$$
\alpha_1(s) = M^s - M' \quad \alpha_2(s) = m' - m^s.
$$

(1)

By assuming that $s$ in the middle point in a luminance transition, the relationship $\alpha_1(s) = \alpha_2(s)$ holds.

In the case of noise, the change in the sign of the difference function expressed as:

$$
D(s) = \alpha_1(s) - \alpha_2(s)
$$

(2)

is a more effective indicator of the presence of a contour [7]. In order to clarify the meaning of $D(s)$, in [7] it is shown that $D(s) = \partial I/\partial n^2$, that is, $D(s)$ approximates the directional derivative of the luminance signal $I$ along the gradient direction. This means that the relationship $D(s) = 0$ is equivalent to find the flex points of luminance transitions. In particular, we will look for zero-points and zero-crossing
points of $D(s)$. Hence the introduction of a threshold is required, so that samples $s$ satisfy the condition:

$$\text{threshold} < D(s) < \text{threshold}. \quad (3)$$

Successively, edge samples $s$ are detected according to the following simple algorithm [7]:

- for each $s = (x_0, y_0)$ so that (3) is satisfied,
- if $D(s) = 0 \iff s$ is edge
- elseif $D(s) \geq 0$ and
  $$(D(x_0-1,y_0) < 0 \text{ or } D(x_0+1,y_0) < 0) \text{ or } (D(x_0,y_0-1) < 0 \text{ or } D(x_0,y_0+1) < 0) \iff s$ is edge.

In other words, at first a preliminary selection of samples that are candidate edge pixels is carried out. Then, by applying the above algorithm to the sample itself and to the four neighboring samples along the horizontal and vertical directions, a first edge detection is achieved. Unfortunately, zeros of the function $D(s)$ are not only flex points of luminance transitions, but also the set of pixels having a neighborhood where luminance is almost constant, and noise causes small fluctuations [7]. These fluctuations may result in a lot of changes in the sign of $D$ that would be incorrectly assumed as edge points. Therefore, in order to better select the previously detected edges, it is necessary to derive from $D(s)$ a matrix (called $P(s)$) that can highlight the discontinuity areas [7]. Details about $P(s)$ will be given in Section IV.

### III. Low-pass Filtering

Before illustrating the first block of the proposed segmentation algorithm (see Fig.1(a)), we would point out that through the paper we adopt the well-known CNN standard model reported in [8]. Moreover, we remind that the CNNUM incorporates advanced computational capabilities [5], including: i) addition and subtraction operations on grayscale images; ii) capability to combine two binary images through boolean operators such as AND, OR; iii) capability to select which cells are going to be processed.

Referring to the low-pass filtering, in order to remove noise patterns, the following templates have to be used for two times:

$$A = \begin{bmatrix} 0 & 1 & 0 \\ -4 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0.1 & 0.1 & 0.1 \\ 0.1 & 0.2 & 0.1 \\ 0.1 & 0.1 & 0.1 \end{bmatrix} \quad I = 0 \quad (4)$$

where $a=0.77$ in the first filtering operation whereas $a=0.25$ in the second one. Differently from the template reported in [2], templates (4) satisfy the condition of having element values less than 3. Successively, the objective is to maintain the sharpness of the scene’s contours by means of a threshold gradient operation [2]. Herein the nonlinear threshold gradient template suggested in [2] has been replaced by a sequence of linear templates. Namely, the approach is based on eight linear templates in the eight directions $N$, $NW$, $NE$, $W$, $E$, $SW$, $SE$ and $S$. For example, in the NW direction the templates are:

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2.94 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 2.94 & 0 & 0 \\ 0 & -2.94 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad I = -0.05 \quad (5)$$

The sequence output is the low-pass (LP) filtered frame $Y_{iLP}$.

### IV. Edge Extraction

The block diagram of the algorithm is shown in Fig. 1(b). In the following, the description of each block is reported.

#### A. Generation of the Matrix $D(s)$

The input of the algorithm (i.e., $Y_{iLP}$) should be processed by applying a routine mainly based on the nonlinear threshold gradient template reported in [2]. Instead of applying templates unsuitable for the CNNUM, herein a threshold gradient algorithm is proposed, which enables the nonlinear template in [2] to be replaced by a sequence of linear templates. The approach is based on eight linear templates, applied in the eight directions $N$, $NW$, $NE$, $W$, $E$, $SW$, $SE$ and $S$. For example, in the NW direction the templates are:

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -3 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad I = -0.05 \quad (6)$$

The remaining templates can be easily derived from (6).

![Fig. 1. Block diagrams: overall algorithm (a); edge extraction algorithm (b).](Image)

In order to obtain the matrix $D(s)$ defined by (2), at first it is necessary to generate two images, which contain concentric circular windows of radius $r$ and $R$, respectively (Fig.2).

![Fig. 2. Grids of circular windows: radius $r$ (left) and radius $R$ (right).](Image)

Successively, it is necessary to compute from the initial image the four values specified in Section II, that is: $M^R$, $m^R$, $M'$ and $m'$. By starting from the top (left corner), these values have to be computed by means of proper max and min linear templates, which are not available in literature. Referring to the computation of $M^R$ and $M'$, the proposed technique for obtaining the max templates is based on two steps. The first phase consists in the consecutive execution of eight rotated difference templates:


\[
A_i = 0 \quad B_i = \begin{bmatrix}
1 & 0 & 0 \\
0 & -1 & 0 \\
0 & 0 & 0
\end{bmatrix} \quad \ldots \quad A_k = 0 \quad B_k = \begin{bmatrix}
0 & 0 & 0 \\
1 & -1 & 0 \\
0 & 0 & 0
\end{bmatrix}
\]  

(\text{all with } i=0), \text{ where the resulting image is the sum of the intermediate results. In this way, all those pixels whose value is larger than zero in the pixel neighborhood are detected. In the second phase this image is supplied as bias map to the template } A \text{ having the central element equal to 1 and the remaining elements zero (along with } B = 0 \text{ and } i=0). \]

Referring to the computation of \( m^s \) and \( m^c \), a similar process is used in order to realize the \( m \) templates. The only difference regards the eight \textit{rotated difference} templates used in the first phase, described by:

\[
A_i = 0 \quad B_i = \begin{bmatrix}
-1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 0
\end{bmatrix} \quad \ldots \quad A_k = 0 \quad B_k = \begin{bmatrix}
0 & 0 & 0 \\
-1 & 1 & 0 \\
0 & 0 & 0
\end{bmatrix}
\]  

(\text{all with } i=0). \text{ Then, by computing the difference described by (1)-(2), a partial matrix } D(s) \text{ is obtained. Namely, in order to find the final matrix } D(s), \text{ it is necessary to iterate the previous operations. This can be done by moving the initial image, along with those containing the two grids of circular windows, using the \textit{right} and \textit{down} templates reported in [8].}

\begin{figure}[h]
\centering
\includegraphics[width=0.4\textwidth]{image3.png}
\caption{Foreman: (a) third frame; (b) image representing the matrix } D(s).\end{figure}

Then, all the partial results are combined until the bottom (right corner) is reached. The final result is the image representing the overall matrix } D(s). \text{ Figure 3(a) shows the third frame of } \textit{Foreman} \text{ video sequence, whereas Fig.3(b) represents the final matrix } D(s).  

\subsection{Edge Detection}

The block diagram for the \textit{edge detection} is shown in Fig.4(a).

\begin{figure}[h]
\centering
\includegraphics[width=0.4\textwidth]{image4.png}
\caption{Edge detection: (a) block diagram; (b) result of the algorithm.}\end{figure}

By exploiting the CNN paradigm, it is at first necessary to satisfy (3). \text{ Then, edge samples are detected according to the algorithm described in Section II. In order to satisfy (3), the \textit{threshold} template reported in [8] is applied, along with the operations \textit{inversion}, \textit{OR} and \textit{inversion} again (via the CNNUM). The next step is to implement the algorithm in Section II by exploiting CNN-based approach. By using the \textit{increase} template reported in [9], the conditions } D(x, y, z) < 0 \text{ \text{ or } D(x, y, z) < 0 \text{ are satisfied, whereas the AND operation enables the constraint } D(s) \geq 0 \text{ to be satisfied. Then, by exploiting the OR operation, the image shown in Fig.4(b) is obtained.}

\subsection{Edge Selection}

In order to better select the previously detected edges, it is necessary to derive from } D(s) \text{ another matrix, called } P(s), \text{ which can highlight the discontinuity areas. The final result will be an image including all the contours selected by the gradient operation. The proposed CNN algorithm works as follows. At first the matrix } D(s) \text{ is processed by means of the } \textit{threshold} \text{ template reported in [8], with } i=0. \text{ Then, the pixels in } D(s) \text{ that correspond to } D(s)>0 \text{ assume the values } M^s, \text{ whereas the pixels in } D(s) \text{ that correspond to } D(s)<0 \text{ assume the values } m^s. \text{ The resulting two images are indicated with } D(M^s) \text{ and } D(m^s), \text{ respectively. Finally, by means of the \textit{zero*} template reported in [5], the two images } D(M^s) \text{ and } D(m^s) \text{ are combined with the result of the threshold operation. The resulting image } P(s) \text{ is reported in Fig.5(a). The corresponding mask, obtained by applying the sequence of linear templates (6) with } I=-0.05, \text{ is reported in Fig.5(b).}

\begin{figure}[h]
\centering
\includegraphics[width=0.4\textwidth]{image5.png}
\caption{Image representing } P(s)(a); \text{ binary mask of } P(s)(b).\end{figure}

Then, in order to delete the open lines, the following \textit{kill open line} template is applied to the mask of } P(s):

\[
A = \begin{bmatrix}
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1
\end{bmatrix} \quad B = \begin{bmatrix}
0 & 2.5 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0
\end{bmatrix} \quad I = -1.
\]

(9)

Successively, the \textit{hollow} template in [2] enables the holes located inside the contours to be filled. Finally, by combining the output of the \textit{hollow} with the output of the edge detection algorithm (see Fig.4(b)) via the \textit{zero*} template reported in [5], it is possible to obtain the image reported in Fig.6(a), which includes all the contours selected by the gradient operation.

\subsection{Edge Cleaning}

In order to reduce all the contours to one-pixel thin lines, the \textit{skeletonization} process is applied. The proposed approach consists of eight linear templates (in the eight directions). For example, in the NW direction the templates are:

\[
A = \begin{bmatrix}
0 & 0 & 0 \\
0 & 3 & 0 \\
0 & 0 & 0
\end{bmatrix} \quad B = \begin{bmatrix}
0.25 & -0.25 & 0 \\
0.25 & -0.25 & 0 \\
0 & -0.25 & 0
\end{bmatrix} \quad I = -0.75.
\]  

(10)
Then, in order to complete open contours and make them closed, the following sequence of complete templates is first proposed:

\[
B_1 = \begin{bmatrix}
1 & -1/7 & -1/7 \\
1/7 & 1/7 & -1/7 \\
-1/7 & -1/7 & 1/7 \\
\end{bmatrix}
\]

with \(A_1 = 0, \ I_1 = 0, \ A_2 = 0 \) and \( I_2 = 0 \). Successively, the following sequence of close templates is suggested:

\[
B_3 = \begin{bmatrix}
0 & 0 & I \\
0 & 1 & 0 \\
I & -1 & -1 \\
\end{bmatrix}
\]

(VW) \[12\]

The last step consists in deleting the remaining open lines. By applying a routine mainly based on the template (9), the image \(Y_{cont}\) showing the final contours is obtained (Fig.6(b)).

### V. MOTION DETECTION

By exploiting the computational capabilities of the CNNUM, the difference between the filtered frames \(Y_{LP}^i\) and \(Y_{LP}^{i-3}\) is computed. Then, differently from [2], the absolute value is carried out using the following linear templates:

\[
A = \begin{bmatrix}
0 & 0 & 0 \\
0 & -1 & 0 \\
0 & 0 & 0 \\
\end{bmatrix}
\]

\[
B = \begin{bmatrix}
0 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 0 \\
\end{bmatrix}
\]

with \(I_{cont} = 0, \ I_{mask} = 1, \ A_1 = 0 \) and \( I_1 = 0 \). The applications to the remaining directions enable the open contours to be "completed and closed".

### VI. OBJECT EXTRACTION

The block diagram of the CNN algorithm for object extraction is shown in Fig.7(b). In the following, each block is described.

#### A. Fragment Extraction

By considering the input frame \(Y_{cont}\), the steps reported in the block diagram in Fig.8(a) have to be applied (that is, the increase template [9], the subtraction, the increase template again, the hollow template [2] and the object-detection template [8]). The output is the binary image containing the fragments (i.e., representing a very small objects extraction).

#### B. Rough Extraction of Objects

The block diagram is reported in Fig.8(b). By considering as input image the previous output of the hollow operation in Fig.8(a), it is necessary to apply the hollow template [2] and the threshold template [8]. In particular, the first one enables the detected objects to be filled, whereas the second one enables the contours to be improved. Then, by applying the object-detection template [8], the current image is compared with the negative of \(Y_{cont}\). By subtracting (via the CNNUM) the current output of the object-detection with the image containing the objects detected at the previous iteration of the algorithm, it is possible to obtain the image shown in Fig.8(c). Since the resulting image is usually constituted by groups of objects, it is necessary to apply a fine object extraction.

#### C. Fine Extraction of Objects

This algorithm is very similar to the previous one, since it is based on the same templates (see Fig.8(b)). The only difference is that the transient time related to the application of the hollow template has to be shorter, so that the filling process is slower than the previous one. This enables more accurate object extractions to be obtained (see Fig.8(d)).

#### D. Combination of Objects and Fragments

Once objects have been found, they have to be combined with the fragments. By considering, for example, the object reported in Fig.9(a), it has to be enlarged by using the increase template in [9]. Successively, the AND between the object in Fig.9(a) and the mask of the extracted fragment is carried out. Finally, by applying the selected objects
extraction template [9] and the OR operation, the image that combines object and fragment is obtained (Fig.9(b)).

![Fig.9](image)

**E. Check on the Moving Objects**

The AND operation is carried out between the result of the previous combination (Fig.9(b)) and the motion detection mask (Fig.7(a)). This enables to check if the result of the previous combination belongs (or not) to the moving objects. The result of the check is shown in Fig.9(c). When all the objects have been checked, the segmentation algorithm gives as final result the image reported in Fig.10(a). The comparison with the method developed in [1]-[2] (see Fig.10(b)) gives a first insight into the capabilities of the proposed approach.

![Fig.10](image)

**VII. DISCUSSION AND CONCLUSION**

Some comparisons are now carried out. Referring to Car-phone video sequence, the comparison between the proposed approach and the one developed in [1]-[2] is shown in Fig.11. The results (see Figs. 10 and 11) highlight the effectiveness of the proposed technique, which provides more accurate segmented objects than the ones obtained by the approach in [1]-[2]. Similar results have been obtained for other benchmark video sequences. In particular, visual inspection clearly shows that, by adopting the proposed algorithm, detected contours are always single-pixel and are much more close to the real contours, with respect to the one proposed in [1]-[2]. This is a key point of our approach, as it is able to effectively detect closed regions with uniform luminance level, which is of paramount importance when dealing with object-oriented coding and object recognition [7]. Now an estimation of the processing time of the proposed segmentation algorithm is given. Referring to Car-phone video sequence, the results are summarized in Table I, where the processing times of each CNN algorithm (time in µs) have been provided. By taking $\tau = 280$ns [9], the estimated execution time of the overall algorithm is 10.99ms. This leads to a processing rate of about 91 frame/s, which exceeds the usual video frame frequency of 30 frame/s. Finally, we would remark that a comparison between the performances of some typical DSP processors and the CNNUM-chip architecture has been carried out in [3]. In particular, referring to the segmentation algorithm illustrated in [3], it can be deduced that the CNNUM is about 10 times faster than the PentiumII@400MHz & MMX™, and about 25 times faster than the TMS320C80 40MHz. These results, along with those related to the segmentation algorithm tested on the CNNUM hardware (see reference [4]), seem to confirm that the CNNUM is a promising architecture for real-time object-oriented video processing applications.

In conclusion, by exploiting rigorous model of the image contours, we have developed a new CNN-based object-oriented segmentation algorithm. The approach, besides taking into account the hardware characteristics imposed by the CNNUM, yields accurate results (better than the ones achievable by other CNN-based techniques) with a video processing rate of about 91 frame/s. We plan to test the overall algorithm on the CNNUM hardware in the near future.

![Fig.11](image)

**TABLE I**

<table>
<thead>
<tr>
<th>Process</th>
<th>Time (µs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low-pass filtering</td>
<td>771 µs</td>
</tr>
<tr>
<td>Edge extraction</td>
<td>10716 µs</td>
</tr>
<tr>
<td>Motion detection</td>
<td>51 µs</td>
</tr>
<tr>
<td>Object extraction</td>
<td>27727 µs</td>
</tr>
<tr>
<td>Total</td>
<td>39265 µs</td>
</tr>
</tbody>
</table>

**REFERENCES**


