# NONLINEAR IMAGE PROCESSING THROUGH SEQUENCES OF FAST CELLULAR NEURAL NETWORKS (FCNN)

*M. Coli [1], P.Palazzari [2], R.Rughi [1]*

[1] Electronic Engineering Department, University 'La Sapienza', Via Eudossiana, 18 - 00184 Rome (Italy)
E-mail coli@die.ing.uniroma1.it, rodolfo.rughi@infoservice.it
[2] ENEA – HPCN Project – C.R. Casaccia, Via Anguillarese, 301, S.P. 100
00060 S. Maria di Galeria (Rome) – E-mail palazzari@casaccia.enea.it

## ABSTRACT

Cellular Neural Networks (CNN) are a powerful paradigm to perform fast nonlinear image processing. They are not widely used because 1) they must be implemented on analogic HW 2) they are not easily programmable and 3) they do not have a fast and robust learning algorithm. In this paper we introduce a new class of CNN, the Fast CNN (FCNN), which can be efficiently implemented on digital HW, are easily programmable and have fast and robust learning algorithms. We give a theoretical description of FCNN and we introduce a fast learning algorithm, based on Simulated Annealing. An image elaboration tool, based on the Learning (LM) and Elaboration (EM) Modules is described. Through an example (texture recognition) we show how FCNN can be used to efficiently implement nonlinear image processing.

## 1. INTRODUCTION

Cellular Neural Networks (CNN) [1],[2] were introduced by L.O. Chua and are a powerful computational paradigm to perform not linear image processing. The universal CNN machine [3] was developed as an analogic programmable environment performing fast processing: computational time is the time elapsed from initial state to the transient completion. An image is transformed through a CNN programmed by a sequence of cloning templates, being cloning template the set of weights connecting neurons and determining the elaboration. CNN universal machine offers an high throughput, due to the high speed of analogic devices, but presents many severe drawbacks which make it inadequate for a widespread diffusion and use. In fact, CNN universal machine is based on a not standard analogic HW; furthermore, like CNNs, it suffers of the lack of fast and reliable learning algorithms.

In order to exploit CNN flexibility without occurring in the above mentioned problems, we developed the Fast CNNs (FCNNs). FCNNs are characterized by

1. toroidal topology
2. discrete time evolution
3. the using of the few steps of the dynamic sufficient to perform the desired transformation (typically 1 to 5 steps).

Because of point 1 a compact and useful representation of FCNN state evolution can be given.

Point 2 allows implementation of FCNN on digital devices. This type of implementation exploits both programmability and high level design tools.

Point 3 is fundamental for the using of powerful and robust heuristics in the learning phase.

Similarly to the universal CNN machine, FCNNs are the kernel for an environment dedicated to image processing and composed by two modules: the first module (Learning Module, LM) performs FCNN learning, determining the cloning template which allows the best approximation (according to the euclidean norm) of the desired transformation. The second module (Elaboration Module, EM) uses a sequence of FCNNs (determined through LM) to perform a complex image transformation; each FCNN, by means of its state evolution, implements an elementary transformation. The environment we developed uses the following steps:

- a complex transformation (C) is decomposed into a sequence of elementary transformations (E); a transformation is specified by the pair <input image – output image>;
- FCNN, which implements E, is determined by LM whenever the cloning template is not yet available;
- EM performs C by sequentially transforming input image through Es. EM uses a library of Es (dynamically upgraded through LM) implemented in an optimized way onto a (eventually specialized) digital computer.

LM is based on powerful heuristics like Simulated Annealing (SA) [4] or Simulated Tempering (ST) [5]; such techniques allow a deep exploration of the searching space but need a very large number of error function evaluations. So, in the case of CNNs, they cannot be adopted due to the very long times needed to compute the error function (the complete

transient must be simulated [6]); consequently, CNN use simpler but less robust learning algorithms (like back-propagation [7]). On the contrary, thanks to previous point 3, FCNNs give their output within few steps of dynamic evolution, allowing a fast evaluation of error function. SA or ST can be so used without occurring in very long learning times (a typical learning operation requires nearly 1 to 5 minutes on a Pentium II machine).

In this work, after an introductory explanation about FCNNs and their learning algorithms, we show how they can be used to perform complex transformations on images; a transformation is obtained through a sequence of elementary steps, each one implemented by a FCNN whose cloning template is determined on the basis of the desired elaboration.

## 2. FCNN THEORY

FCNNs are characterized by a bidimensional toroidal topology, i.e. neurons are defined over an (*m* x *n*) grid with connections between corresponding neurons on opposite borders. Given two points $p_i=(x_{i1},x_{i2})$ (i=1,2), we consider the case of an (*m* x *m*) grid and we define the distance between $p_1$ and $p_2$ as:

$$D(p_1,p_2)=\min\begin{cases}\max\limits_{i=1,2}\left(|x_{1i}-x_{2i}|\right)\\ m-\max\limits_{i=1,2}\left(|x_{1i}-x_{2i}|\right)\end{cases}$$

On a FCNN, the neighborhood with radius *r* of a neuron *p* is the set $N_r(p_i)$ of all the neurons at distance less or equal to *r*, i.e.

$$N_r(p_i)=\left\{p\,|\,p_i,\ p\in R, D(p_i,p)\le r\right\}$$

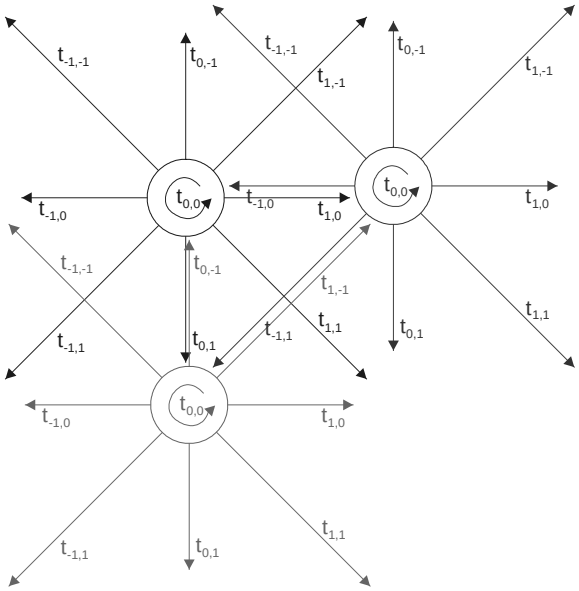
Neuron with coordinates (i,j) is connected to neuron (k,l) if (k,l) belongs to the neighborhood of (i,j). The weight connecting the two neurons is $t_{(i,j)\to(k,l)}$.

As in classical CNN, each neuron is connected to its neighborhood by the same set of weights. Because of its spatial invariance, the set is called Cloning Template and is defined as

$$CT=\left\{t_{(i,j)\to(k,l)}\,|\,(k,l)\in N_r(i,j)\right\}$$

For instance, in the case of *CT* with radius *r*=1, neuron (i,j) is connected to itself by the weight $t_{(i,j)\to(i,j)}$ and to the nearest neighbor neurons on the North, South, East, West, NE, NW, SE, SW directions by the weights $t_{(i,j)\to(i+1,j)}$, $t_{(i,j)\to(i-1,j)}$, $t_{(i,j)\to(i,j+1)}$, $t_{(i,j)\to(i,j-1)}$, $t_{(i,j)\to(i+1,j+1)}$, $t_{(i,j)\to(i+1,j-1)}$, $t_{(i,j)\to(i-1,j+1)}$, $t_{(i,j)\to(i-1,j-1)}$.

The following figure shows the interactions among a neuron and its east and south neighborhoods.




As we see, each neuron interacts with the neurons within its neighborhood through the same set of weights, the Cloning Template (CT).

*CT* fully determinates the elaboration performed by a FCNN. In fact, indicating with $s_{i,j}(n)$ the state of neuron (i,j) at the discrete time instant *n*, the successive state is derived through the state evolution equation

$$s_{i,j}(n+1)=\sum_{(k,l)\in N_r(i,j)} t_{(i,j)\to(k,l)}\cdot s_{k,l}(n) \qquad (1)$$

The output of a FCNN is determined by the sign of the first derivative of the state, that is

$$y_{ij}(n+1)=\begin{cases}+1 & s_{ij}(n+1)>s_{ij}(n)\\ y_{ij}(n) & s_{ij}(n+1)=s_{ij}(n)\\ -1 & s_{ij}(n+1)<s_{ij}(n)\end{cases} \qquad (2)$$

Frequently it is useful to use FCNNs which elaborate only neurons with initial state greater than 0, that is

$$s_{i,j}(n+1)=\begin{cases}\sum\limits_{(k,l)\in N_r(i,j)} t_{(i,j)\to(k,l)}\cdot s_{k,l} & s_{i,j}>0\\ 0 & otherwise\end{cases} \qquad (3)$$

we call them Active FCNN (AFCNN) because they elaborate only neurons not in the off state.

We introduce now a compact representation of FCNN, based on matrices.

A radius *r* cloning template is expressed through the weight matrix **t**:

$$t\left((2r+1);(2r+1)\right)=\begin{bmatrix} t_{-r,-r} & \cdots & t_{0,-r} & \cdots & t_{+r,-r}\\ & & \vdots & & \\ t_{-r,0} & \cdots & t_{0,0} & \cdots & t_{+r,0}\\ & & \vdots & & \\ t_{-r,+r} & \cdots & t_{0,+r} & \cdots & t_{+r,+r}\end{bmatrix} \qquad (4)$$

Let us consider a 4×4 FCNN with cloning template with radius $r$=1:

$$\mathbf{t}(3;3)=\begin{bmatrix} t_{-1,-1} & t_{0,-1} & t_{1,-1} \\ t_{-1,0} & t_{0,0} & t_{1,0} \\ t_{-1,1} & t_{0,1} & t_{1,1} \end{bmatrix}=\begin{bmatrix} t_0 & t_1 & t_2 \\ t_3 & t_4 & t_5 \\ t_6 & t_7 & t_8 \end{bmatrix}$$

As FCNN is 4×4, let us consider a 4×4 block right circulant matrix RP, i.e a matrix having 4×4 blocks as entries, each row being obtained as a rotation toward right of the previous row, i.e.

$$\mathbf{RP}=\begin{bmatrix} \mathbf{Rp}_1 & \mathbf{Rp}_2 & \mathbf{Rp}_3 & \mathbf{Rp}_4 \\ \mathbf{Rp}_4 & \mathbf{Rp}_1 & \mathbf{Rp}_2 & \mathbf{Rp}_3 \\ \mathbf{Rp}_3 & \mathbf{Rp}_4 & \mathbf{Rp}_1 & \mathbf{Rp}_2 \\ \mathbf{Rp}_2 & \mathbf{Rp}_3 & \mathbf{Rp}_4 & \mathbf{Rp}_1 \end{bmatrix} \qquad (5)$$

Block entries $\mathbf{Rp}_i$ (i=1,…,4) are still 4×4 scalar right circulant matrices, defined through the **t** weights:

$$\mathbf{Rp}_1=\begin{bmatrix} t_4 & t_5 & 0 & t_3 \\ t_3 & t_4 & t_5 & 0 \\ 0 & t_3 & t_4 & t_5 \\ t_5 & 0 & t_3 & t_4 \end{bmatrix},\ \mathbf{Rp}_2=\begin{bmatrix} t_7 & t_8 & 0 & t_6 \\ t_6 & t_7 & t_8 & 0 \\ 0 & t_6 & t_7 & t_8 \\ t_8 & 0 & t_6 & t_7 \end{bmatrix},$$

$$\mathbf{Rp}_3=\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix},\ \mathbf{Rp}_4=\begin{bmatrix} t_1 & t_2 & 0 & t_0 \\ t_0 & t_1 & t_2 & 0 \\ 0 & t_0 & t_1 & t_2 \\ t_2 & 0 & t_0 & t_1 \end{bmatrix}$$

$\mathbf{Rp}_1$, $\mathbf{Rp}_2$, $\mathbf{Rp}_4$, are the right circulant matrices associated to the three rows of the cloning template $(t_3 \quad t_4 \quad t_5)$, $(t_6 \quad t_7 \quad t_8)$, $(t_0 \quad t_1 \quad t_2)$ and obtained by inserting a zero in the 3rd position; $\mathbf{Rp}_3$ is the null 4×4 matrix.

FCNN state at (discrete) time n can be represented through the column vector

$$\mathbf{s}(n)=[\mathbf{s}_1(n) \quad \mathbf{s}_2(n) \quad \mathbf{s}_3(n) \quad \mathbf{s}_4(n)]^T,$$

whose i[th] entry (i=1,2,…,n) is the column vector

$$\mathbf{s}_i(n)=[s_{i,1}(n) \ \ s_{i,2}(n) \ \ s_{i,3}(n) \ \ s_{i,4}(n)]^T;$$

so we can think the bidimensional state of an $m$×$m$ FCNN as an $m^2$ column vector, obtained by row-wise representing the state matrix.

On the base of previous definitions, it is easy to verify that evolution of FCNN state from time n to (n+1) can be compactly written as:

$$\begin{bmatrix} \mathbf{s}_1(n+1) \\ \mathbf{s}_2(n+1) \\ \mathbf{s}_3(n+1) \\ \mathbf{s}_4(n+1) \end{bmatrix}=\begin{bmatrix} \mathbf{Rp}_1 & \mathbf{Rp}_2 & \mathbf{Rp}_3 & \mathbf{Rp}_4 \\ \mathbf{Rp}_4 & \mathbf{Rp}_1 & \mathbf{Rp}_2 & \mathbf{Rp}_3 \\ \mathbf{Rp}_3 & \mathbf{Rp}_4 & \mathbf{Rp}_1 & \mathbf{Rp}_2 \\ \mathbf{Rp}_2 & \mathbf{Rp}_3 & \mathbf{Rp}_4 & \mathbf{Rp}_1 \end{bmatrix}\cdot\begin{bmatrix} \mathbf{s}_1(n) \\ \mathbf{s}_2(n) \\ \mathbf{s}_3(n) \\ \mathbf{s}_4(n) \end{bmatrix} \qquad (6)$$

In the general case of an $m$×$m$ FCNN, we indicate with $\mathbf{Rp}_i$ the $m$×$m$ right circulant matrix associated to the (i-r+1)[th] row of the radius $r$ cloning template **t** (eq. 4) extended through the insertion of zeroes into the $r$+2,…,$m$-$r$ positions:

$$\mathbf{Rp}_i=\begin{bmatrix} t_{0,k} & t_{1,k} & \cdots & t_{r,k} & 0 & \cdots & t_{-r,k} & \cdots & t_{-1,k} \\ t_{-1,k} & t_{0,k} & \cdots & \cdots & t_{r,k} & 0 & \cdots & \cdots & t_{-2,k} \\ & & & & \vdots & & & & \\ t_{-r,k} & \cdots & t_{0,k} & \cdots & \cdots & t_{r,k} & 0 & 0 & 0 \\ 0 & t_{-r,k} & \cdots & t_{0,k} & \cdots & \cdots & t_{r,k} & 0 & 0 \\ & & & & \vdots & & & & \\ t_{1,k} & \cdots & t_{r,k} & 0 & \cdots & 0 & t_{-r,k} & \cdots & t_{0,k} \end{bmatrix}$$

$$\text{being } \begin{cases} \text{i} = 1,...,\text{r}+1 \\ \text{k} = \text{i}-1 \end{cases} \text{ and } \begin{cases} \text{i} = \text{M}-\text{r}+1,...,\text{M} \\ \text{k} = \text{i}-\text{M}-1 \end{cases} \qquad (7)$$

and $\mathbf{Rp}_i$ is the null $m$×$m$ matrix when $r$+2≤ i ≤$m$-$r$.

Now we are able to write eq. 6 for the general case of an $m$×$m$ FCNN having cloning template with radius $r$:

$$\begin{bmatrix} \mathbf{s}_1(n+1) \\ \mathbf{s}_2(n+1) \\ \vdots \\ \mathbf{s}_M(n+1) \end{bmatrix}=\begin{bmatrix} \mathbf{Rp}_1 & \mathbf{Rp}_2 & \cdots & \mathbf{Rp}_M \\ \mathbf{Rp}_M & \mathbf{Rp}_1 & \cdots & \mathbf{Rp}_{M-1} \\ \vdots & & & \vdots \\ \mathbf{Rp}_2 & \cdots & \mathbf{Rp}_M & \mathbf{Rp}_1 \end{bmatrix}\cdot\begin{bmatrix} \mathbf{s}_1(n) \\ \mathbf{s}_2(n) \\ \vdots \\ \mathbf{s}_M(n) \end{bmatrix} \qquad (8)$$

With obvious extension of notations to the general case, we can write one step evolution of FCNN state as:

$$\mathbf{s}(n+1)=\mathbf{RP}\cdot\mathbf{s}(n) \qquad (9)$$

Because of associative property of matrix product, evolution for m instants of the FCNN state is compactly given by:

$$\mathbf{s}(n+m)=\mathbf{RP}^m\cdot\mathbf{s}(n) \qquad (10)$$

## 3. THE LEARNING AND THE ELABORATION MODULES (LM, EM)

An elementary transformation E is specified through the pair <initial image $I_i$ - output image $I_o$>.

An FCNN is characterized by

- the radius $r$ of CT and
- the number of simulation steps $N_s$.

Depending on the locality of the elaboration to be executed, $N_s$ varies from 1 to 5; for a given problem, the larger is $r$, the smaller is $N_s$. Le us consider a computation in which a neuron changes its state depending on the state of a neuron at distance d: such a computation can be obtained through an FCNN having radius $r$=d/k and $N_s$=k (k=1,2,…). The increasing of k causes a reduction of $r$ but also of the transformation accuracy: so we search for a trade-off between speed (small $r$ and large $N_s$) and accuracy (large $r$ and small $N_s$) of the elaboration. The increasing of k moves the propagation of the influence of a neuron from space to time.

We indicate with $I(CT,I_i,N_s)$ the image obtained by elaborating for $N_s$ steps $I_i$ through the FCNN with cloning template $CT$.

Once specified E=<$I_i$,$I_o$>, we chose a couple <$r$, $N_s$> and, using these values, LM searches for the CT* which gives the best approximation of the E transformation. In order to find the desired CT, LM must solve the following minimization problem:

$$\left\| I_o - I\left(CT^*, I_i, Ns\right)\right\| = \min_{CT}\left(\left\| I_o - I\left(CT, I_i, Ns\right)\right\|\right) \quad (11)$$

Solution of the previous problem gives the $CT^*$ which allows the best approximation of the desired $I_o$.

We nearly solved problem (11) through either Simulated Annealing(SA) [4] or Simulated Tempering (ST) [5] algorithms; they both are heuristics based on Monte Carlo simulations at several temperatures, corresponding to situations in between the melted and frozen states. SA and ST algorithms use the New(CT) function which returns a CT' slight differing from CT (for example one or two weights are varied of a small percentage of their original value). An outline of the SA algorithm is the following, where the cooling schedule (i.e. the starting temperature $t_0$, the cooling parameter k<1 and the *frozen* condition) is determined as reported in [8] (ST is easily derivable from SA with little modifications).

**SA algorithm**
**Input**
$r$, $N_s$, $I_i$, $I_o$;
**Output**
CT* so that $\left\| I_o - I\left(CT^*, I_i, Ns\right)\right\| \le \left(\left\| I_o - I\left(CT, I_i, Ns\right)\right\|\right)$

**Begin**
$t$:=$t_0$
cost:= $\left\| I_o - I\left(CT, I_i, Ns\right)\right\|$
**while** not *frozen*
**for** $i$=1 **to** $k$ **do**
CT'=New($CT$);
costnew= $\left\| I_o - I\left(CT', I_i, Ns\right)\right\|$;
accept=0,
if (costnew<cost) accept=1;
else if exp((cost-costnew)/t)>p accept=1;
if accept=1 C$T$=C$T$'; cost=costnew;
**end for**
$t$:=k*$t$;
**end while**
**end**

Previous algorithm requires nearly from 1 to 5 minutes to determine $CT$ on a Pentium II machine.

Once obtained CT performing E, LM inserts it into a library (CT_LIB) containing all the CTs available.

The Elaboration Module (EM) consists of an optimized implementation of FCNN (and AFCNN). EM is programmed by sequentially calling the cloning template performing a given transformation and specifying its input and output images. The composition of all these transformations performs the desired complex transformation C=<$I_i$,$I_o$>.

A typical program for EM is

**Begin**

CT_identifier_1($I_{i1}$,$I_{o1}$)/OP_identifier($I_{i1a}$,$I_{i1b}$,$I_{o1}$);

…

CT_identifier_k($I_{ik}$,$I_{ok}$)/OP_identifier($I_{ika}$,$I_{ikb}$,$I_{ok}$);

**end**

where CT_identifier is one of the CTs contained in the CT_LIB and OP_identifier denotes one of the operators (which implement the functionality clearly specified by its name) MASK_AND, MASK_OR, MASK_XOR, IMG_DIFF, IMG_ADD, IMG_NOT

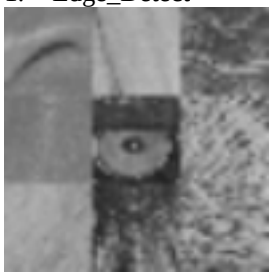
## 4. EXAMPLE: TEXTURE RECOGNITION

In this section we show how EM can be used to extract uniform and texture areas from an image. We use the following CTs:

- Edge_Detect,
- Empty_Homogeneous_Areas,
- Fill_Texture_Areas.

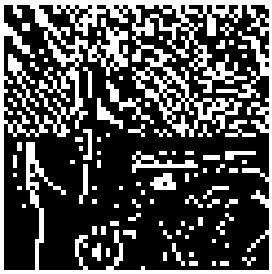
For each CT we report the pair <$I_i$,$I_o$> describing the elaboration, the cloning template obtained through LM and the corresponding output image.

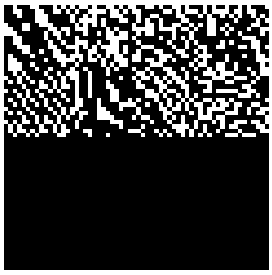
1. Edge_Detect

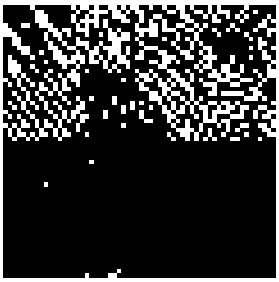


| | | |
|---|---|---|
| -0.10 | 0.34 | 0.12 |
| 0.57 | **-1.00** | 0.39 |
| 0.07 | 0.42 | 0.02 |

2. Empty_Homogeneous_Areas

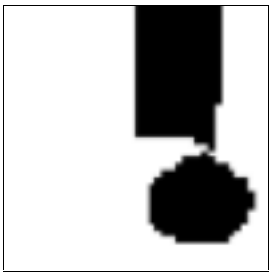






| | | | | |
|---|---|---|---|---|
| 0.45 | 0.87 | 0.33 | 0.64 | 0.67 |
| 0.96 | 0.34 | -0.59 | -0.11 | 0.40 |
| 0.42 | -0.03 | **-0.92** | -0.25 | -0.08 |
| 1.00 | 0.41 | 0.17 | 0.55 | 0.79 |
| 0.78 | 0.17 | -0.15 | -0.11 | 0.57 |

3. Fill_Texture_Areas

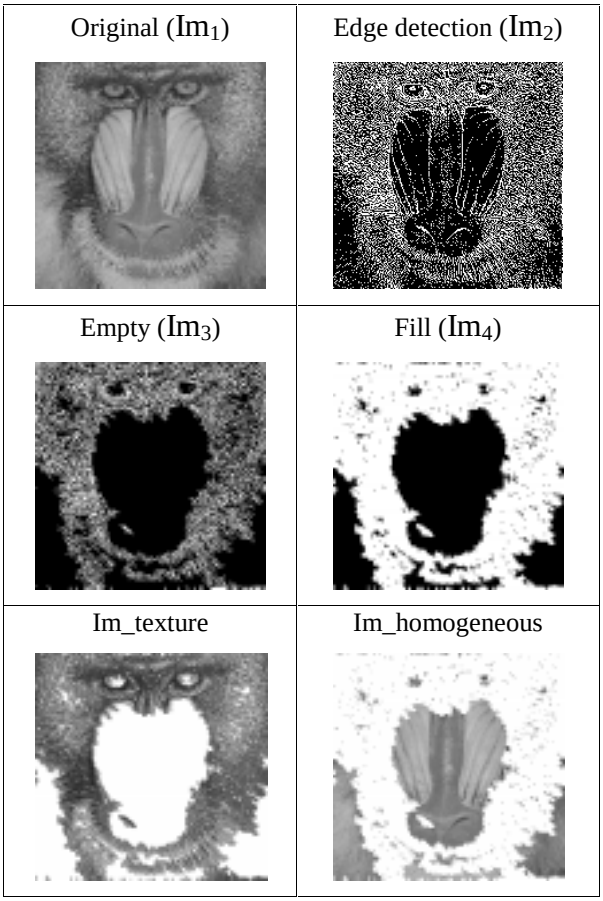


| | | | | |
|---|---|---|---|---|
| 0.40 | -1.00 | -0.63 | -0.50 | -0.26 |
| 1.00 | 0.79 | 0.91 | 0.97 | 0.89 |
| -0.46 | 0.55 | **0.29** | 0.01 | -0.01 |
| 0.45 | 1.00 | 1.00 | 0.98 | 0.85 |
| 0.85 | 0.38 | 0.87 | -0.17 | -0.07 |

The EM program to detect homogeneous and texture areas is the following:

**begin**
Edge_Detect($Im_1$,$Im_2$);
Empty_Homogeneous_Areas($Im_2$,$Im_3$);
Fill_Texture_Areas($Im_3$,$Im_4$);
MASK_OR(($Im_1$,$Im_4$, Im_homogeneous);
IMG_NOT($Im_4$,$Im_5$);
MASK_OR(($Im_1$,$Im_5$, Im_texture);
**end**

In sequent figures we show the input image (baboon) and the intermediate and output images of previous program.




## 5. REFERENCES


[1] L.O. Chua, L. Yang, “Cellular Neural Networks: Theory”, IEEE Trans. on CAS, vol. 35, 1257-1272 (1988)

[2] L.O. Chua, L. Yang, “Cellular Neural Networks: Application”, IEEE Trans. on CAS, vol. 35, 1273-1290 (1988)

[3] T. Roska, L.O. Chua “The CNN Universal Machine: an analogic array computer”, IEEE Trans. on CAS-II, march 1993.

[4] S. Kirkpatrick, C. Gelatt, Jr., and M. Vecchi, "Optimization by Simulated Annealing" Science, Vol. 220, No. 4598. pp. 498-516, May 1983

[5] E. Marinari, G. Parisi, “Simulated Tempering: a new Monte Carlo scheme”, hep-lat/9205018

[6] T. Kozek, T. Roska, O. Chua, “Genetic Algorithm for CNN Template Learning”, IEEE Tr. CAS, vol. 40, no 6, pp 392-402, june 1993

[7] Balsi, M. “Recurrent back-propgation for CNN”, H. Dedieu (Ed.), ECCTD’93, Elsveier Science Publisher, Amsterdam, 1993, 677-682.

[8] Deckers A., Aarts E., "Global Optimization and Simulated Annealing", *Mathematical Programming*, Vol. 50, 1991.