Color image processing in a cellular neural-network environment

Color Image Processing in a Cellular Neural-Network Environment

Chi-Chien Lee and Jose Pineda de Gyvez, *Member, IEEE*

Abstract— **When low-level hardware simulations of cellular neural networks (CNN's) are very costly for exploring new applications, the use of a behavioral simulator becomes indispensable. This paper presents a software prototype capable of performing image processing applications using CNN's. The software is based on a CNN multilayer structure in which each primary color is assigned to a unique layer. This allows an added flexibility as different processing applications can be performed in parallel. To be able to handle a full range of color tones, two novel color mapping schemes were derived. In the proposed schemes the color information is obtained from the cell's state rather than from its output. This modification is necessary because for many templates CNN has only binary stable outputs from which only either a fully saturated or a black color can be obtained. Additionally, a postprocessor capable of performing pixelwise logical operations among color layers was developed to enhance the results obtained from CNN. Examples in the areas of medical image processing, image restoration, and weather forecasting are provided to demonstrate the robustness of the software and the vast potential of CNN.**

I. Introduction

THE cellular neural network (CNN) paradigm has rapidly evolved to cover a wide range of applications which are typically characterized by their spatial dynamics [1]. One particular area of big interest is filtering for image processing. Enormous advances have been made by many researchers in this field [2]. Among the relevant work is the creation of templates capable of several image processing applications and the development of special purpose algorithms such as halftoning and character recognition, to mention some [3]–[6]. However, except for the fundamental work on color processing developed by Roska *et al.* [7], all of the work presented so far deals only with black and white images. This probably stems from the fact that the simpler CNN system results in binary states, giving up, somehow, the wide range that the activation energy is capable of. Yet, to handle realistic situations it is necessary to advance the state of the art into color image processing.

Another interesting and propitious area of research concerns multilayer CNN [8]. Simulation strategies based on CNN multilayer architectures are an ideal vehicle for color image processing. This is because each pixel's color can be handled as a triplet ⟨red green blue⟩ (RGB) whose combinations yield a secondary color. It follows then that it is possible to allocate a layer of CNN cells to each primary color component, carry out the processing independently, and then form the triplet to see the results. We call this approach a *sequential color processing* mode. The counterpart is a *concurrent color processing* mode. This mode results from applications in which it is not desired to split a pixel's color into its three basic components. In this case, it would be necessary to have an output function that would be based on the Euclidean distance, or any other norm, of the three RGB values.

Worth emphasizing is that while software prototypes prove the potential of CNN, a great deal of research has been advocated to hardware implementations which can be used for on-line applications in real time [9]. It is a fact that the local interconnectivity properties of CNN make it very attractive for very large scale integration (VLSI) implementations. Unfortunately, low-level hardware simulations are very costly and a behavioral simulator, such as the one hereby presented, becomes necessary to explore new applications. Unlike advanced image processing software which is optimized for application specific cases, e.g., edge detection, this software emulates behaviorally the hardware properties of multilayer CNN architectures. The particular architecture described here with outputs at the cell's state is especially suitable for applications in color image processing. The basic structure of the simulator is based on a high-performance software capable of efficiently dealing with large images in the order of 10^5 pixels [10], [11]. The simulator operates in a *sequential* mode. This provides an added flexibility to create individual templates that can be applied on single colors to obtain a full mix of applications/colors. The simulator runs in an X-Windows environment and uses standard graphics formats as input. It allows to edit images and several CNN control parameters among other features.

A preliminary background on CNN is introduced in Section II. Sections III and IV address the behavioral simulation approach and color processing capabilities of our software. Section V presents the software environment and postprocessing capabilities for image processing. Section VI acquaints us with some image processing applications of CNN in practical situations and also presents comparisons between different color-mapping strategies implemented in the simulator.

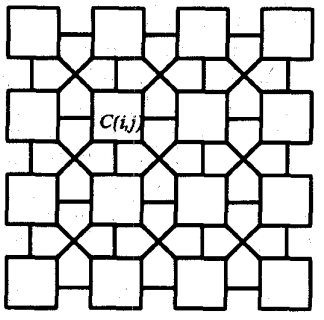
II. Cellular Neural Networks

Cellular automata is distinguished mainly because its graph follows a regular lattice Z. Its neighborhood structure and the transition function among vertices are translation invariant, i.e., they are the same for all vertices; additionally, the state

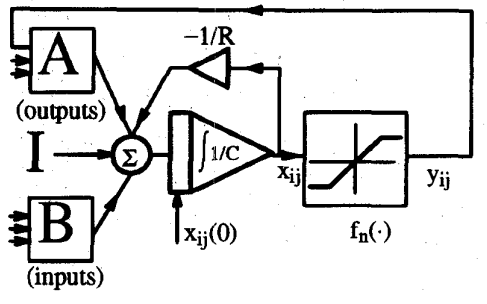
Manuscript received July 8, 1994; revised February 15, 1995 and May 25, 1995. This work was supported by the Office of Naval Research under Grant N00014-91-1-0516.

The authors are with the Department of Electrical Engineering, Texas A&M University, College Station, TX 77843 USA.

Publisher Item Identifier S 1045-9227(96)02883-4.



(a)



(b)

Fig. 1. Cellular neural networks: (a) array structure and (b) block diagram of one cell.

updating rule is synchronous. Let $I = \mathbf{Z}^d$, where d is the dimension of the lattice. If a set of connections $V \subset \mathbf{Z}^d \times \mathbf{Z}^d$ is translation invariant, meaning $(j, i) \in V$ iff $(j+k, i+k) \in V$, the graph $G = (\mathbf{Z}^d, V)$ is called a cellular space. Cellular automata are automata defined on the cellular space whose transition function is also translation invariant: $f_i = f$ for any $i \in \mathbf{Z}^d$ with f: $Q^{|V|} \rightarrow Q$, and Q the set of states.

Consider an $M \times N$ CNN having $M \times N$ cells arranged in M rows and N columns, see Fig. 1(a). The basic unit of a CNN is called a cell [13], [14]. Any cell on the ith row and jth column, $C(i, j)$, is connected only to its neighbor cells, i.e., adjacent cells interact directly with each other. This *neighborhood* is denoted as $N(i, j)$. Cells not in the immediate neighborhood have indirect effect because of the propagation effects of the dynamics of the network. Each cell has a state x, a constant external input u, and output y. The equivalent block diagram of a continuous-time cell is shown in Fig. 1(b). The first-order nonlinear differential equation defining the dynamics of a cellular neural network cell can be written as follows:

$$C\frac{dx_{ij}(t)}{dt} = -\frac{1}{R}x_{ij}(t) + \sum_{C(k,l)\in N_r(i,j)} A(i, j; k, l)y_{kl}(t) + \sum_{C(k,l)\in N_r(i,j)} B(i, j; k, l)u_{kl} + I \quad (1)$$

$$y_{ij}(t) = \tfrac{1}{2}(|x_{ij}(t)+1| - |x_{ij}(t)-1|) \quad (2)$$

where x_{ij} is the state of cell $C(i, j)$, $x_{ij}(0)$ is the initial condition of the cell, C and R conform the integration time constant of the system, and I is an independent bias constant. In an actual circuit implementation, the integration process is not instantaneous and will depend on a time constant $\tau = RC$. $A(i, j; k, l)$ and $B(i, j; k, l)$ are space-invariant programming templates for all cells $C(k, l)$ in the neighborhood $N(i, j)$ of cell $C(i, j)$; u_{kl} represents the external input and y_{ij} represents the output equation, i.e., the activation function for the cell. This function is shown in Fig. 2. The state of each cell is bounded for all time $t > 0$ and, after the transient has settled down, a class of cellular neural networks always approaches one of its stable equilibrium points [13]. This last fact is relevant because it implies that the system will not oscillate. Furthermore, if the system satisfies that the center element of template A is greater than one, i.e., $A_{ii} > 1$, then the settled state values will converge to absolute values greater than one, i.e., $|x_{ij}| \geq 1$ [13].

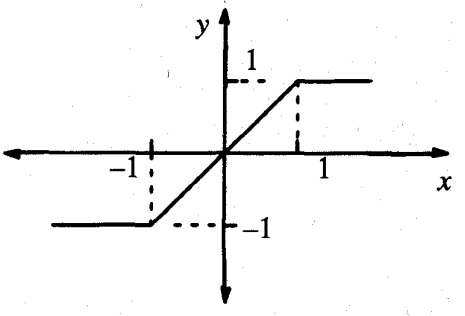


Fig. 2. CNN's output function.

Notice from the summation operators that each cell is affected by its neighbor cells. $A(\cdot)$ acts on the output of neighboring cells and is referred to as the *feedback operator*. $B(\cdot)$ in turn affects the input control and is referred to as the *control operator*. Specific entry values of matrices $A(\cdot)$ and $B(\cdot)$ are application dependent and space invariant. The matrices are also known as *cloning templates* [1], [15], [16]. A constant bias I and the cloning templates determine the *transient behavior* of the cellular nonlinear network.

In image processing applications the concept of locality is important. Usually, a pixel's value is calculated based only on its neighbor pixels. Neural networks like Hopfield lack this property of locality making them unsuitable for image processing applications.

III. Image-Based Behavioral Simulation

To see why cellular neural networks can be used for image processing let us first approximate the differential equation (1a) by a difference equation. Let $t = \tau n$, where τ is a constant time step, and let us approximate the derivative of x_{ij} by its corresponding difference equation. After rearranging terms, the corresponding difference equation of (1) is

$$x_{ij}(\tau n + 1) = -x_{ij}(\tau n) + \frac{\tau}{C}\left[\frac{-1}{R}x_{ij}(\tau n) + \sum_{C(k,l)\in N_r(i,j)} A(i, j; k, l)y_{kl}(\tau n) + \sum_{C(k,l)\in N_r(i,j)} B(k, j; k, l)u_{kl} + I\right]. \quad (3)$$

Algorithm: (Multi–Layer Raster CNN simulation)

Obtain the input image, initial conditions and templates from user;
/* M,N = # of rows/columns of the image */
while (converged_cells < total # of cells) {
  for (layer=0; layer < 3; layer++) {
    for (i=1; i<=M; i++)
      for (j=1; j<=N; j++) {
        if (convergence_flag[layer][i][j])
          continue; /* current cell already converged */
        /* calculation of the next state*/

$$x_{layer,ij}(t_{n+1}) = x_{ij}(t_n) + \int_{t_n}^{t_{n+1}} f'(x(t_n))\ dt$$

        /* convergence criteria */
        if ($\frac{dx_{layer,ij}(t_n)}{dt} = 0$ and $y_{layer,kl} = \pm\ 1,\ \forall\ C_{layer}(k,l) \in N_r(i,j)$) {
          convergence_flag[layer][i][j] = 1;
          converged_cells++ ;
          }
        } /* end **for** */
      /* update the state values of the whole image*/
      for (i=1; i<=M; i++)
        for (j=1; j<=N; j++) {
          if (convergence_flag[layer][i][j]) continue;
          $x_{layer,ij}(t_n) = x_{layer,ij}(t_{n+1})$;
        }
      #_of_iteration++;
  }
} /* end **while** */

Fig. 3. Behavioral level algorithm for raster image processing using CNN.

Equation (3) can be interpreted as a two-dimensional filter for transforming an image's pixel represented by $x_{ij}(\tau n)$, into another one represented by $x_{ij}(\tau n + 1)$ [14]. In other words, (3) represents an image at time $n\tau$ which depends on the initial image $x_{ij}(0)$ and the dynamic rules of the cellular neural network. The filter is nonlinear because of the nonlinear output function of CNN. For the one-step filter in (3), the pixel values, $x_{ij}(\tau n+1)$, of an image are determined directly from the pixel values, $x_{ij}(\tau n)$, in the corresponding neighborhood $N(i, j)$. Therefore, a one-step filter can only make use of local properties of images. When global properties of an image are important, the above one-step filter can be iterated n times to extract additional information from the image. Observe in general that this interpretation implies that each pixel is mapped onto a CNN cell. That is, we have an image processing function in the spatial domain that can be expressed as $g(x, y) = T[f(x, y)]$ where $f(\cdot)$ is the input image, $g(\cdot)$ the processed image, and T is an operator on $f(\cdot)$ defined over the neighborhood of (x, y). For CNN this means that an output image pixel is only influenced by input image pixels within some extent area r in the neighborhood of the corresponding output image pixel. In common image processing applications $T(\cdot)$ is usually carried out as a convolution process between a response function array and the input image.

Now recall that (1) is space invariant, which means that $A(i, j; k, l) = A(i-k, j-l)$ and $B(i, j; k, l) = B(i-k, j-l)$ for all i, j, k, l. Therefore, the image transformation process can be seen as a scanning procedure in which the pixels are mapped one-to-one to the CNN cells. The basic approach is to imagine a square subimage area centered at (x, y), with the subimage being the same size of the templates involved in the simulation. The center of this subimage is then moved from pixel to pixel starting, say, at the top left corner and applying the A and B templates at each location (x, y) to solve the difference equation. This procedure is repeated for each time step, for all the pixels. We denote an instance of this

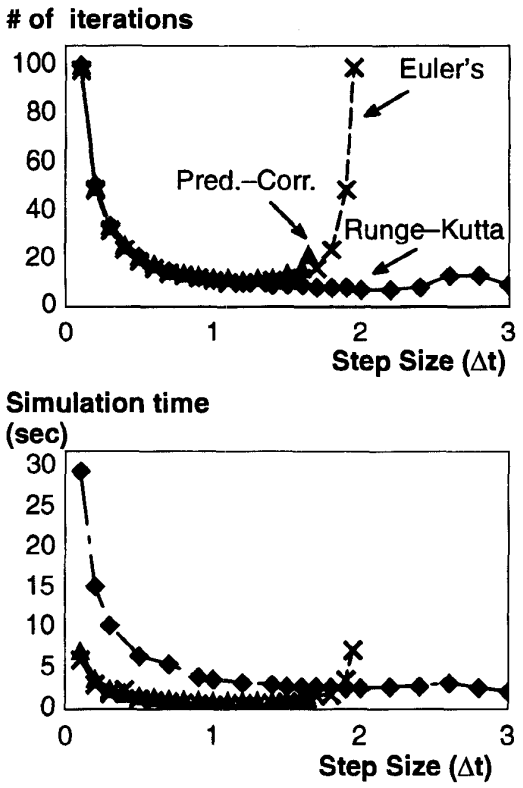


Fig. 4. Performance of the CNN software for number of iterations versus integration time steps and simulation times (SPARC-2) for distinct time steps.

image scanning-processing as an *"iteration."* The processing stops when it is found that the states of all CNN cells have converged to steady-state values, and when the outputs of its neighbor cells are saturated, e.g., they have a ± 1 value. This whole simulating approach is referred to as *raster simulation.* Notice that when the templates are located on the border of the input image, the scanning process does not involve all of its elements. To deal with this border effect a center zero padded superposition model is used. That is to say, a virtual set of border cells, initialized to zero state values, is created. The multilayer CNN raster simulation algorithm is presented in Fig. 3.

For the purpose of solving the initial-value problem, well established single-step methods of numerical integration techniques are used [19]. Three of the most widely used single-step algorithms are applied in the CNN behavioral simulator described here. They are the Euler's algorithm, the improved Euler predictor-corrector algorithm, and the fourth-order (quartic) Runge–Kutta algorithm. Euler's method is the simplest of all algorithms for solving ODE's. It is an explicit formula which uses the Taylor-series expansion to calculate the approximation. The improved Euler predictor-corrector method uses both explicit (predictor) and implicit (corrector) formulas. The integral is calculated by multiplying a step size τ with the averaged sum of both the derivative of the discretized state, $x(n\tau)$, and the derivative of the predicted state, $x_p[(n+1)\tau]$, at the next time step. The fourth-order Runge–Kutta method is the most costly among the three

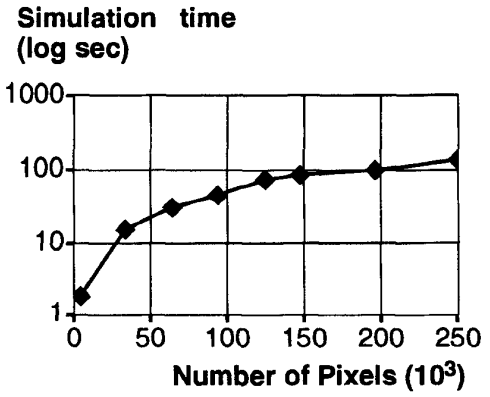


Fig. 5. CPU performance (SPARC-2) for distinct image sizes in number of pixels.

methods in terms of computation time, as it requires four derivative evaluations per time step. However, its high cost is compensated by its accuracy in transient behavior analysis.

Since speed is one of the main concerns in the simulation, finding the maximum step size that still yields convergence for a template can be helpful in speeding up the system. The speed-up can be achieved by selecting an appropriate step size τ for that particular template, See Fig. 4(a). The importance of selecting an appropriate τ can be easily visualized in Fig. 4(b). If the step size chosen is too small, the simulation might take many iterations, hence longer time to achieve convergence. On the other hand, if the step size taken is too large, the simulation might not converge at all or it would converge to erroneous steady-state values; the latter remark can be observed for the Euler integration method in the plots of Fig. 4(b). The past results were obtained by simulating a small image of size $16\times$ 16 (256 pixels) using an edge detection template on a diamond figure on only one layer. In Fig. 5, simulation time computations using an averaging template for images of sizes to about 250 000 pixels are shown; the hardware platform is a SPARC-2 workstation.

IV. Color Processing

To perform any kind of color image processing, a color model must be selected. With CNN, there is no exception. The purpose of a color model is to facilitate the specification of colors in some standard manner. Basically, a color model is a specification of a three-dimensional coordinate system and a subspace within that system where each color is represented by a point [17], [18].

Virtually, all computer display hardware employs the RGB model. In this scheme a color is represented by the relative amounts of color (intensities) of three primary colors that are required to produce the given color. The RGB model is an *additive primary system* that describes a color in terms of the percentage of red, green, and blue in the color. These three colors are called *additive primaries.* Mixing them is like combining colored lights: combining 100% red, 100% green, and 100% blue creates white, i.e., ⟨255, 255, 255⟩ in RGB values. Conversely, combining 0% red, 0% green, and 0% blue creates black, ⟨0, 0, 0⟩ in RGB values.

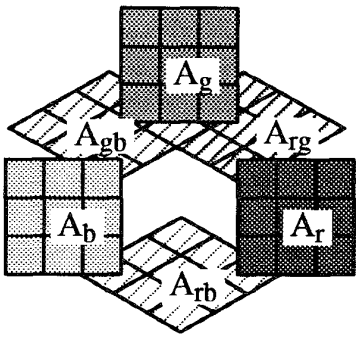


Fig. 6. Organization of templates in the multilayer structure.

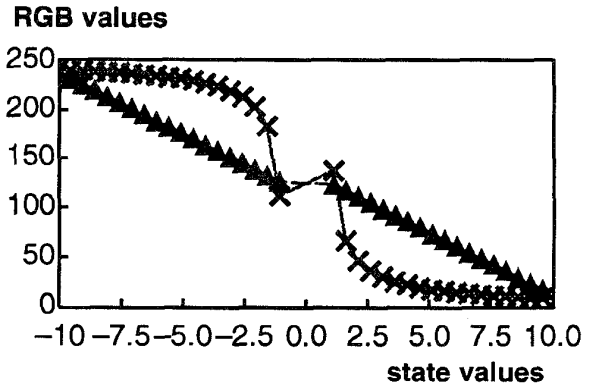


Fig. 7. Color mapping schemes showing RGB values versus state values.

Using the RGB model has the advantage that each primary color can be represented by a CNN layer, e.g., red, green, and blue layers $\mathcal{L}_{\mathrm{R}}$, $\mathcal{L}_{\mathrm{G}}$, and $\mathcal{L}_{\mathrm{B}}$. Thus, a simulation approach is to have the triplet ⟨RGB⟩ processed by a three-layer CNN, with each layer processing a primary color. Following this idea, it is then possible to apply distinct templates to each color layer and even to apply templates in between color layers. Therefore, with the ability to process RGB separately, plus the interlayer template effects, more complex image processing applications can be done. It is thus possible to do, say, edge detection in $\mathcal{L}_{\mathrm{R}}$, and averaging in $\mathcal{L}_{\mathrm{G}}$, simultaneously.

To be able to work with multiple layers, the basic CNN equation (1) can rapidly be expanded to a matrix equation of the following form:

$$\frac{d\boldsymbol{x}_{ij}(t)}{dt} = -\boldsymbol{x}_{ij}(t) + \sum_{C(k,l)\in N_r(i,j)} \underline{\boldsymbol{A}}(i,\, j;\, k,\, l)\boldsymbol{y}_{kl}(t) + \sum_{C(k,l)\in N_r(i,j)} \underline{\boldsymbol{B}}(i,\, j;\, k,\, l)\boldsymbol{u}_{kl} + \boldsymbol{I} \tag{4a}$$

$$\boldsymbol{y}_{ij}(t) = \tfrac{1}{2}(|\boldsymbol{x}_{ij}(t)+1| - |\boldsymbol{x}_{ij}(t)-1|) \tag{4b}$$

where for simplicity the time integration constant has been assumed to be unity. In this last equation, instead of only one state variable per cell there are three state variables to be able to process color. $\underline{\boldsymbol{A}}$ and $\underline{\boldsymbol{B}}$ are block triangular matrices and $\boldsymbol{I}$, $\boldsymbol{x}$, $\boldsymbol{y}$ are vectors as follows:

$$\underline{\boldsymbol{A}} = \begin{bmatrix} A_r & 0 & 0 \\ A_{rg} & A_g & 0 \\ A_{rb} & A_{gb} & A_b \end{bmatrix}$$

$$\underline{\boldsymbol{B}} = \begin{bmatrix} B_r & 0 & 0 \\ B_{gb} & B_g & 0 \\ B_{rb} & B_{gb} & B_b \end{bmatrix} \tag{5a}$$

$$\boldsymbol{x} = \begin{bmatrix} x_{rij} \\ x_{gij} \\ x_{bij} \end{bmatrix}$$

$$\boldsymbol{y} = \begin{bmatrix} y_{rij} \\ y_{gij} \\ y_{bij} \end{bmatrix}$$

$$\boldsymbol{u} = \begin{bmatrix} u_{rij} \\ u_{gij} \\ u_{bij} \end{bmatrix}$$

$$\boldsymbol{I} = \begin{bmatrix} I_{rij} \\ I_{gij} \\ I_{bij} \end{bmatrix} \tag{5b}$$

where subindexes r, g, b have been used to refer to color layers $\mathcal{L}_{\mathrm{R}}$, $\mathcal{L}_{\mathrm{G}}$, and $\mathcal{L}_{\mathrm{B}}$, respectively. Notice that although the state variables are independent of each other, layer interaction is permitted through the A and B templates, see Fig. 6. For instance, template A_{rg} has effect on both red and green layers, simultaneously.

The characteristics generally used to distinguish one color from another are brightness, hue, and saturation. Brightness embodies the chromatic notion of intensity. Hue is an attribute associated with the dominant wavelength in a mixture of light waves. Thus, hue represents a dominant color as perceived by the observer; when an object is called red, orange, or yellow one is specifying its hue. Saturation refers to the relative purity or the amount of white light mixed with a hue. The pure spectrum of colors is fully saturated. Colors such as pink (red and white) are less saturated, with the degree of saturation being inversely proportional to the amount of white light added.

Recall now from (2) that the CNN's stable output value is binary if the center element of template A is greater than one, i.e., $A_{ii} > 1$. In other words, if the color is taken directly from the output function the color would be either fully saturated or black. Moreover, combining the three saturated colors ⟨RGB⟩ would yield only a small gamut of distinct colors. To be able to make use of a full range of hues, we take the color information from the cell's state rather than from the output itself, or, we use templates with $A_{ii} < 1$. Notice however that the cell's state, x, is not bounded to $+1$, and while with fully saturated colors there is a straight mapping from CNN output values to color intensities, e.g., $\mathcal{C}$: $\{-1,\, 1\} \rightarrow \{0,\, 255\}$, the problem here is more complex as the state x can take any value from a larger range. In other words, we need to find a function capable of mapping all real numbers to the closed interval $[0 \cdots 255]$, e.g., $\mathcal{C}$: $\mathbb{R} \rightarrow \{0,\, 1,\, 2,\, \cdots,\, 255\}$. We investigated two color mapping schemes: a *continuous* mode and a *quantized* mode. The latter one is based on a linear mapping using the maximum and minimum layer colors as bounds to generate a discrete (quantized) range of colors. Recall from (2) that valid settled state values in our model exclude the open interval range

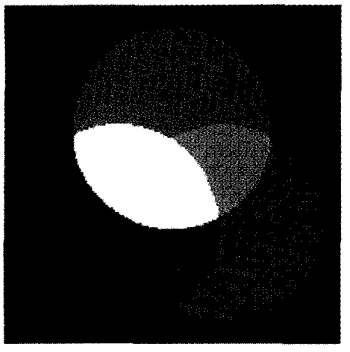


Fig. 8. Benchmark used to measure the effectiveness of the color mapping techniques. All colors are fully saturated.

$(-1, 1)$ for $A_{ii} > 1$. The following quantized mapping was applied for distinct stable states:

$$\begin{aligned} z &= \frac{J}{2} + \frac{J}{2i}\log_2\left\{2^i\left[\frac{x-1}{x+1}\right]\right\} \quad \text{for} \quad x > 1 \\ z &= \frac{J}{2} - \frac{J}{2i}\log_2\left\{2^i\left[\frac{x+1}{x-1}\right]\right\} \quad \text{for} \quad x < -1 \\ z &= 0 \qquad\qquad\qquad\qquad\qquad\quad \text{otherwise} \end{aligned} \tag{6}$$

where J is the absolute maximum color value, i.e., $J = 255$, and i is the number of bits to represent a pixel value. Notice that this value is split at half the color range. This is an arbitrary cutoff which has given us good visual perceptive results.

For the continuous mode mapping, a linear transformation was employed. This transformation can be characterized by the function C: $\mathbb{R} \rightarrow \{-1, 1\}$. In other words, the set of real numbers is mapped to numbers bounded between -1, 1. A second mapping can then be applied to the bounded numbers so that their values can be scaled to the appropriate color range values between 0–255. The transformation looks as follows:

$$\begin{aligned} z &= \frac{J}{2} - \left\{\frac{J}{2}\left[\frac{x-1}{x_U}\right]\right\} \quad \text{for} \quad x > 1 \\ z &= \frac{J}{2} - \left\{\frac{J}{2}\left[\frac{x+1}{x_L}\right]\right\} \quad \text{for} \quad x < -1 \\ z &= 0 \qquad\qquad\qquad\qquad\quad \text{otherwise} \end{aligned} \tag{7}$$

where x_U and x_L are taken as the maximum and minimum state values, respectively, of the entire CNN array at one iteration.

Fig. 7 displays the graphs corresponding to both color mapping techniques. The plots show RGB values versus state values. It can be seen that the continuous mapping approach presents a fairly good distribution of RGB values. The quantized mapping scheme is characterized by the abrupt step between state values of $(-1, 1)$. Notice that for this approach most of the RGB values are concentrated in the upper and lower range. This results in the desired limited set of colors that can be displayed.

Fig. 8 shows the benchmark used to test the mapping techniques. The template used for this example corresponds to a nonfiltering application characterized by minimum feedback and high gain feedforward as follows:

$$\begin{aligned} \boldsymbol{A} &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \\ \boldsymbol{B} &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 200 & 0 \\ 0 & 0 & 0 \end{bmatrix} \\ \boldsymbol{I} &= -1. \end{aligned} \tag{8}$$

This template is applied to all three layers $\mathcal{L}_{\mathrm{R}}$, $\mathcal{L}_{\mathrm{G}}$, and $\mathcal{L}_{\mathrm{B}}$. A high entry value in the B template ensures that the strength of the input pixel value remains unchanged. This is further balanced with I bias whose negative value brightens every single pixel in the whole image. We found no color difference between the benchmark and processed result for the quantized mode. The continuous color mapping was also tested on the benchmark of Fig. 8. This technique presented a small error of 2.3% on each primary color. This difference is actually not perceived by simple visual inspection.

We can conclude that both color mapping techniques are good. From our own experience we have noticed that for applications in which sharp color changes need to be highlighted the quantized mode projects good visualization results. In applications for which the color change is smooth the continuous mode is very suitable. Examples of applications with sharp color changes are edge detection and thresholding, and applications with smooth color changes are averaging and noise removal, to just mention a few.

V. A CNN Post Processor

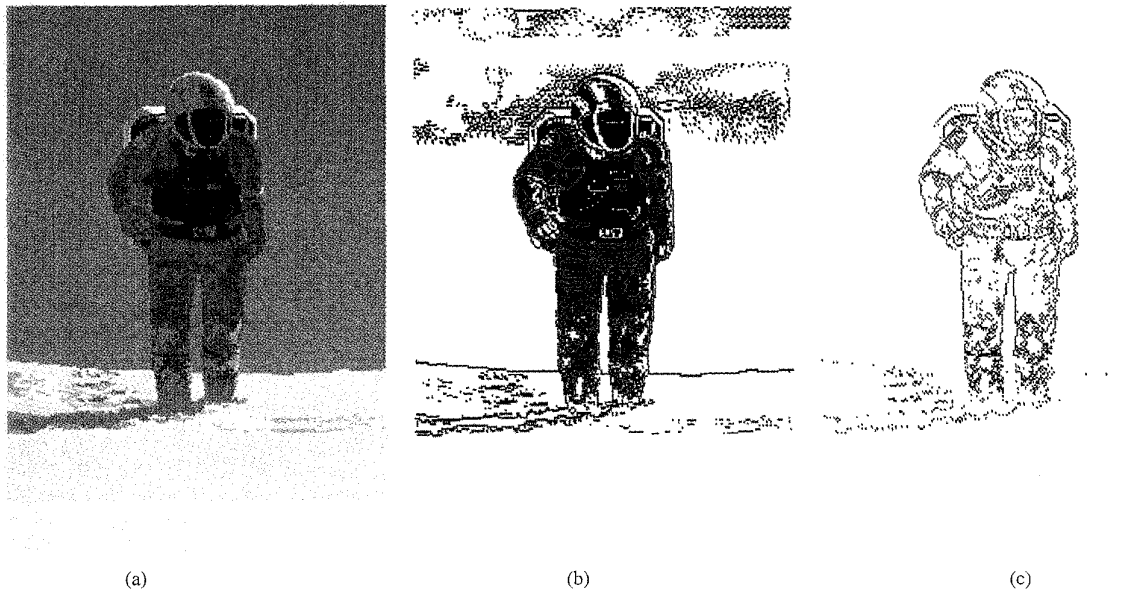
The CNN simulator was built using many features of the public domain software XPaint [20]. The environment is menu driven and allows the user to create color palletes and new canvas, in addition to the standard graphics features such as brushes, lines, circles, etc.

Doing image processing with CNN may not always yield the desired *visual* results and postprocessing becomes then necessary to enhance the visualization of the image. Therefore we developed a CNN postprocessor that consists of a compiler capable of handling logical pixelwise operations among distinct color layers. This compiler follows the trends of having CNN as an analogic microprocessor [12], [21]. The added capability allows us to create new processed images with, for example, one layer processed by CNN and the remaining layers logically manipulated between CNN results and the original image. Detailed examples of this extended processing capability are given in the next section. Fig. 9 shows the syntax of the postprocessing language using a Backus–Naur form notation; keywords and variables are identified as boldface and italic words, respectively.

The following describes the syntax of the postprocessor. All the files to be processed must be specified at the beginning of the program as indicated in statement 1. When a file is read, the program splits the pixel information into its basic RGB components. This strategy is used to create three unique

```
1: main_file      ::= "(" MAIN (files)+ (process)+ (output)+ ")"  .
2: files          ::= NAME .
3: process        ::= "(" (process_descr) ")"  .
4: process_descr  ::= (var) "->" (layer) "," (var) "->" (layer)
                       (operand) (var) "->" layer .
                  |   (var) "->" (layer) "," (var) "->" (layer) .
                  |   (var) "->" (layer) "," (var) "->" (layer)
                       (operand) NUMBER.
                  |   (var) "->" (layer) "," NUMBER (operand) (var)
                      "-" ">" (layer).
                  |   (var) "->" (layer) "," (negation) (var)
"->"                      (layer) .
5: var            ::= NAME .
6: operand        ::= (AND | "&&") .
                  |   (OR  | "||") .
                  |   (XOR | "^") .
                  |   (SL  | "<<") .
                  |   (SR  | ">>") .
7: negation       ::= NOT .
8: layer          ::= RED .
                  |   GREEN  .
                  |   BLUE   .
9: output         ::= "(" OUTPUT "(" (files) "," (var) ")" ")" .
```

Fig. 9. Syntax of the CNN postprocessor.



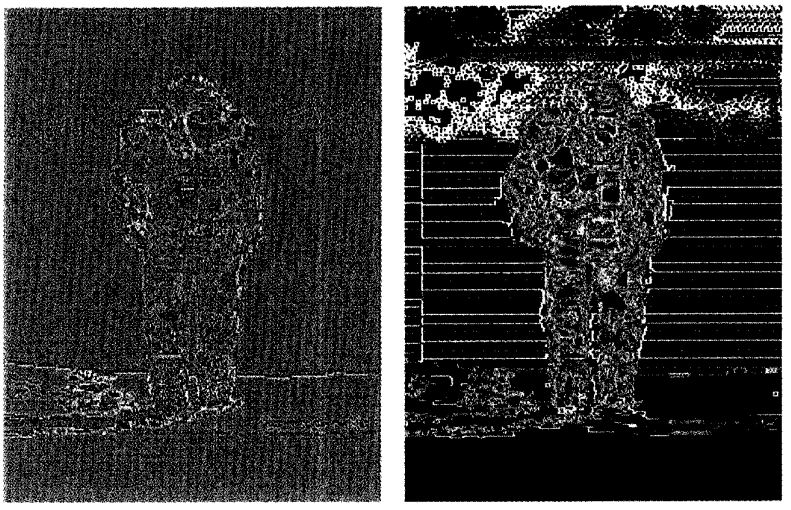
(a) (b) (c)

Fig. 10. (a) Image "Iceworld," (b) contrasting effect, and (c) edge detection.

layers that contain the color-coded information of the image. Statement 3 shows the logical pixelwise operations among layers. These operations include the conventional NOT, OR, AND, XOR, shift-left, and shift-right functions indicated in statement 6. Operations can be performed on the layers of a file or a variable but must always be stored in a variable. The only three valid layers are assigned to the triplet ⟨RGB⟩ and are specified by means of keywords, see statement 8. Every variable's layer is initialized to "black" when first used. Finally, the new processed image is spooled out in statement 9 in which it is required to specify the name of the output file and the variable containing the image to be printed.

VI. Color Image Processing Using CNN

This section of the paper will try to demonstrate the capabilities of our software and the enormous potential that CNN has on a wide variety of fields. For this purpose, we will present three examples with applications in medical image processing, weather forecast, and simple color manipulation. Where possible an explanation on the design of the templates



(d) (e)

Fig. 10. *Continued.* (d) result from Laplacian template, and (e) result from Sobel template.

used in the example-applications will be provided. However, the reader is encouraged to read the corresponding references for full details.

The first example deals with color contrasting. Let us bring your attention to the top region of Fig. 10(a) (160 590 pixels) at the height of the astronaut's helmet. Here, it is very difficult to perceive a set of clouds hidden in the blue background. This image was processed with the following templates [22]:

$$\boldsymbol{A} = \begin{bmatrix} 0.01 & -0.075 & 0.01 \\ -0.075 & 1.28 & -0.075 \\ 0.01 & -0.075 & 0.01 \end{bmatrix}$$

$$\boldsymbol{B} = \begin{bmatrix} -0.04 & -0.13 & -0.04 \\ -0.12 & 0.71 & -0.13 \\ -0.04 & -0.13 & -0.04 \end{bmatrix}$$

$$\boldsymbol{I} = -0.365. \tag{9}$$

The purpose of this template combination is to do a soft edge detection on all three color layers. The simulator was set to operate in a quantized color mode and was stopped after three iterations. The resulting processed image with the clouds uncovered is shown in Fig. 10(b). Fig. 10(c) shows the same image after the edge detection process was completed in 21 iterations. Edging and contrasting operations performed by CNN are quite obvious. This particular example raises the following interesting remark. Notice that although CNN is searching for the steady-state solution of a partial differential equation, in image processing applications intermediate or partial solutions may be sufficient to visualize some interesting results. For instance, in addition to obtaining the edge features of this picture, it was possible to visualize the "hidden" clouds before CNN reached its final solution. As noted in Section III, the image transform by a cellular neural network is a dynamical transform. Thus, it is important to consider the transient response. By taking the information from the state of the cell rather than from its output, a wealth of data can be obtained before the system reaches equilibrium because of the continuous transition of states. Naturally, these data can be meaningful or meaningless depending upon the application.

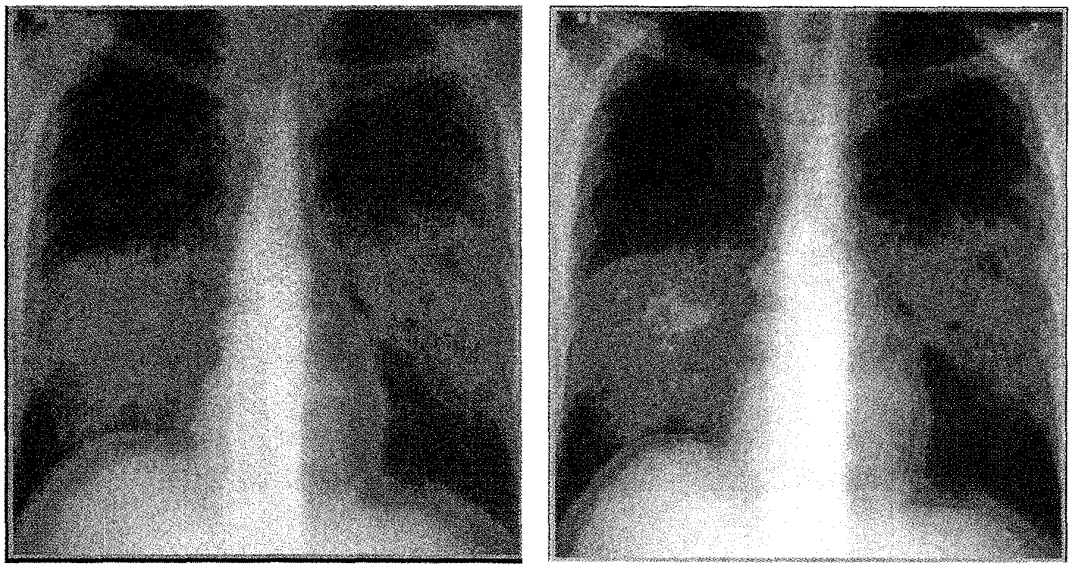
Fig. 10(d) and (e) shows the outcome of applying the Sobel and Laplace edge detection operators using the "image works" tool of a Silicon Graphics workstation. These operators are well established in conventional linear one-step filtering [17], [18]. The application of the operator is equivalent to making every element of the A template be equal to zero and letting the CNN run using only the B template. By doing so, essentially all the state dynamics are cut out of the operation since local feedback interactions are suppressed. Under these conditions, (1) is reduced now to

$$C\frac{dx_{ij}(t)}{dt} = -\frac{1}{R}x_{ij}(t) + \sum_{C(k,l)\in N_r(i,j)} B(i, j; k, l)u_{kl} + I \tag{10a}$$

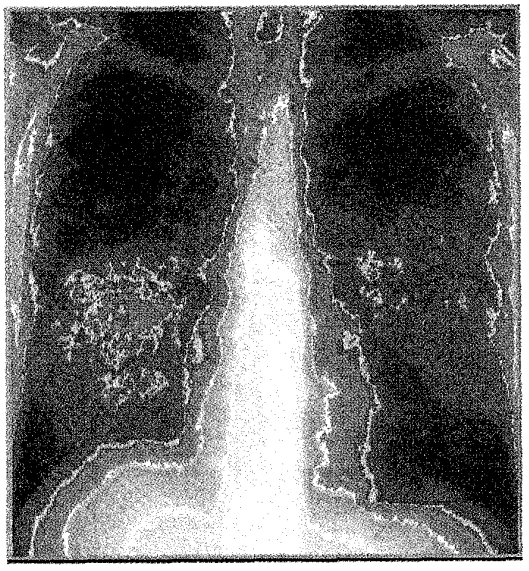
which for steady-state conditions resembles the convolutional operation of one-step image filters [18], see (10b)

$$x_{ij}(t) = \sum_{C(k,l)\in N_r(i,j)} B(i, j; k, l)u_{kl} + I. \tag{10b}$$

Thus, conventional one-step filters are a particular case of CNN. The second example deals with the X-rays image of a chest cage (148 370 pixels) displayed in Fig. 11(a). The objective is to color-code the black and white image and to highlight hidden features in the esternun. The image was treated two times with distinct templates. First, a "pixel peeler" template was used to widen visual and hidden contours in the image. This template was chosen instead of a common edge detector because the latter leaves only the contours and darkens the body of the image. This would actually alter the information contained in the image as our goal was to only highlight the edges. The template "peels" the rightmost pixel from any two or more adjacent pixels; this action is executed



(a) (b)



(c)

Fig. 11. (a) Chest cage X-rays image, (b) after a filler operation, and (c) after using the CNN postprocessor.

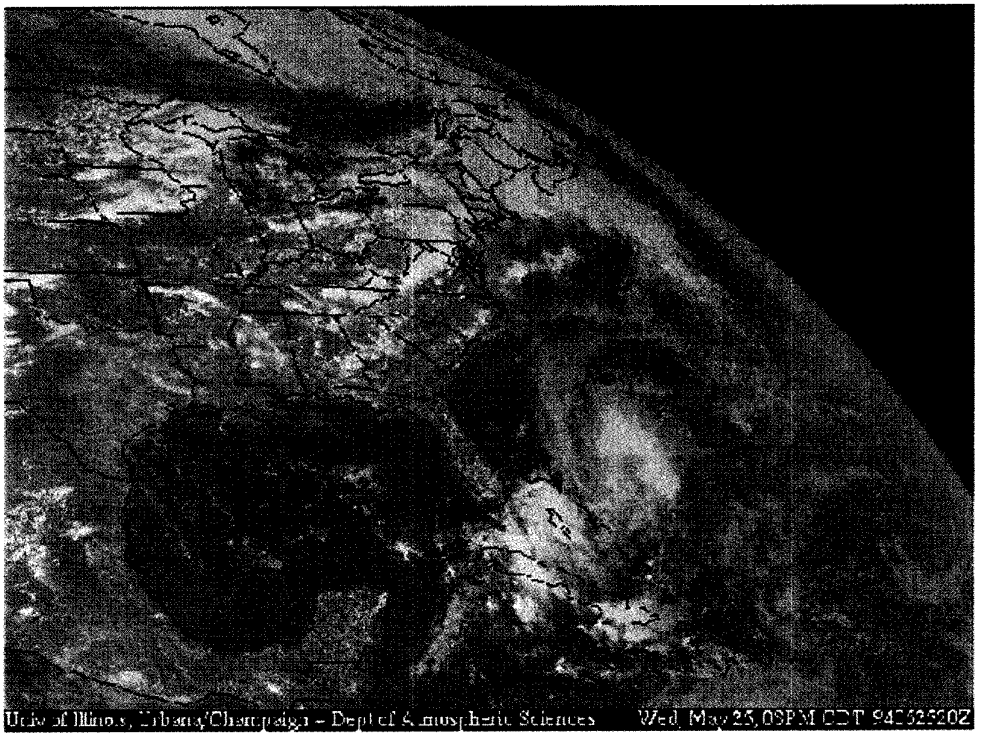
through the B template. Both templates are as follows [23]:

$$\boldsymbol{A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$
$$\boldsymbol{B} = \begin{bmatrix} 0 & 0 & 0 \\ 3 & 3 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$
$$\boldsymbol{I} = -1. \tag{11a}$$

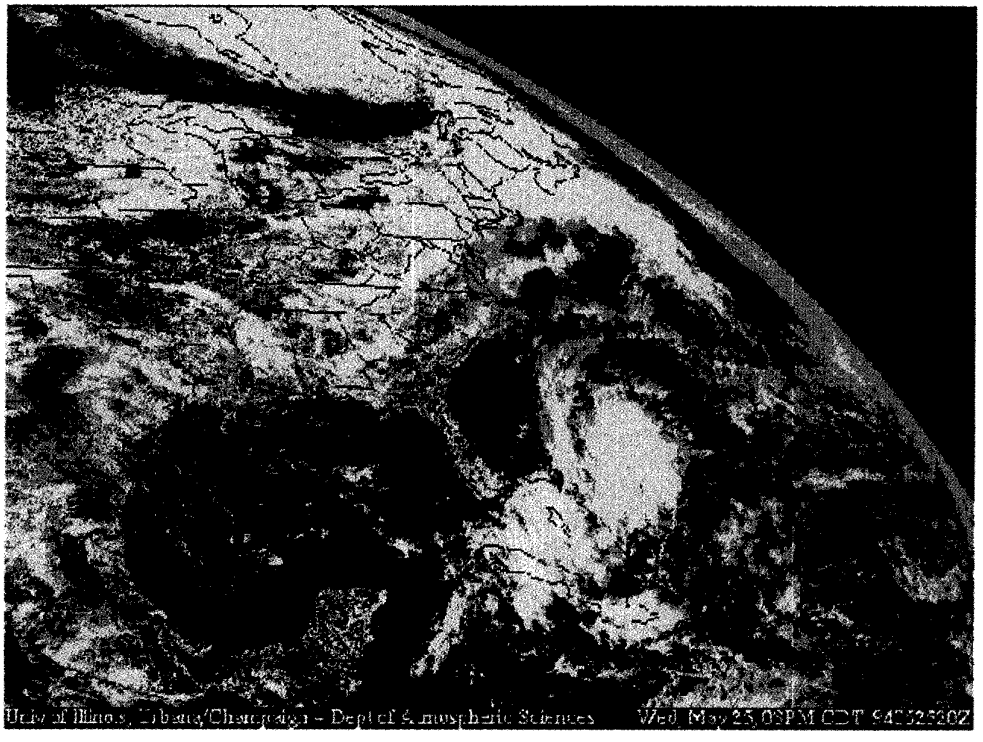
The resulting image was further processed by a "filler template." The template was applied only to the red layer setting the simulator to work in a quantized color mode; the other two remaining layers were processed with the unity template described by (7). These templates are as follows [24]:

$$\boldsymbol{A} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$
$$\boldsymbol{B} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$
$$\boldsymbol{I} = -1. \tag{11b}$$

Basically, the function of the A template is to feed the pixel values of the neighboring pixels into the current pixel. A

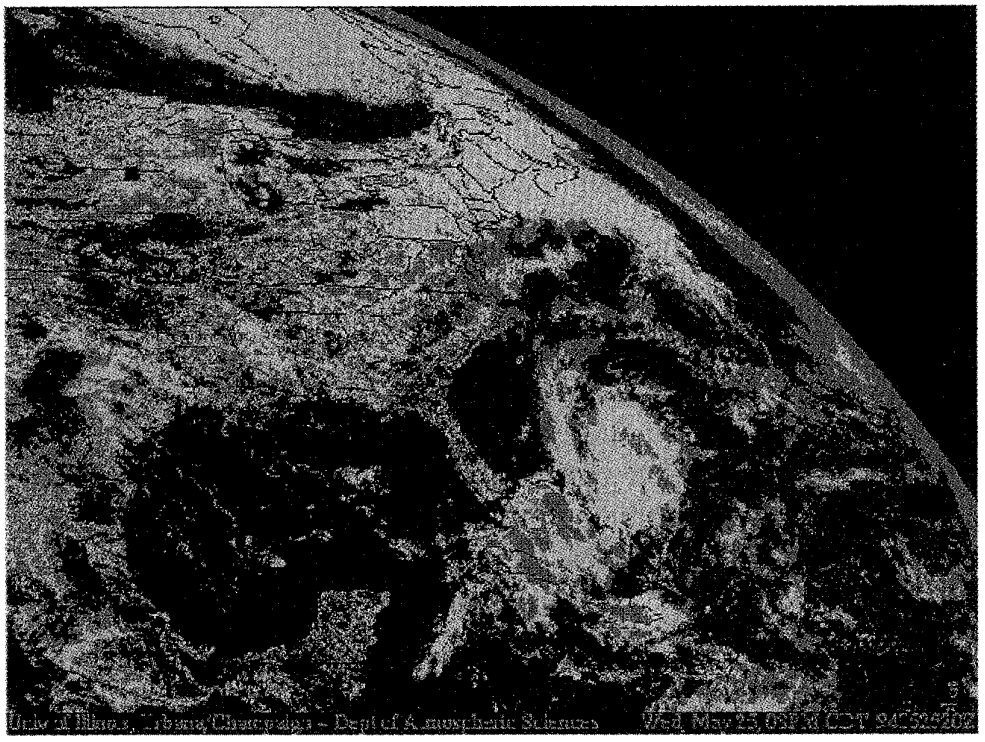


(a)



(b)

Fig. 12. (a) Satellite weather map and (b) after thresholding operations with CNN.



(c)

Fig. 12. *Continued.* (c) Satellite weather map after using the CNN postprocessor.

negative bias is used to avoid having an excessively dark image. The result of both operation is shown in Fig. 11(b). Observe the red dots along the enhanced contour of the ribs. The CNN processor undoubtedly did its work. Unfortunately, the visualization of the results is not optimal as the image has still many gray tones. Hence, the image was postprocessed to achieve an optimal color manipulation. The postprocessing program is listed below.

```
(main original.gif edge.gif edge2.gif
(xx→red, edge.gif→red || edge2.gif→red)
(xx → green, original.gif → green ||
edge.gif → green)
(xx → blue, original.gif → blue)
(output (out,xx)) ).
```

The files `original.gif`, `edge.gif`, and `edge2.gif` correspond to the original black and white image, the image processed in the continuous color mode, and the image processed in the quantized mode, respectively. The program does the following: 1) The red layer of both edge detection operations is ORed to obtain soft and hard tone contours; 2) the green layer of the original image is ORed with the green layer of the edge detection obtained using the quantized color mode; and 3) the blue layer is left intact. The result of this post processing is shown in Fig. 11(c). It is quite obvious that CNN was able to detect the hidden features in the esternun which otherwise are impossible to perceive from the original black and white image.

This forthcoming example demonstrates CNN's ability to color-code a black and white image. The image presented in Fig. 12(a) corresponds to an actual satellite weather map (200 984 pixels) taken on 25 May, 1994 (the reader can see that we had very cloudy weather). The goal was to obtain a result as close as possible to the standard color-coded maps used in weather forecasts. These maps use bright tones of green to show light clouds up to red tones to show strong rain intensities. The image was treated with color thresholding techniques [25] using intensively the bias factor I. The result of this operation is displayed in Fig. 12(b). The templates are as follows:

$$
\begin{aligned}
\boldsymbol{A}_{\mathbf{red}} &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2.9 & 0 \\ 0 & 0 & 0 \end{bmatrix} \\
\boldsymbol{A}_{\mathbf{green}} &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 20.2 & 0 \\ 0 & 0 & 0 \end{bmatrix} \\
\boldsymbol{A}_{\mathbf{blue}} &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \\
\boldsymbol{B} &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \\
I_{\text{red}} &= -0.25 \\
I_{\text{green}} &= -5.9 \\
I_{\text{blue}} &= -0.55.
\end{aligned} \tag{12}
$$

The A template is used basically to operate on the quality of the colors. A large value in the center of this template helps to obtain a brighter color on the result. This explains the very high value for the green layer. A large value in the center of the B template is used to reinforce the effect of the input image on the final result. The thresholding function of I bias works as follows. Negative values, force the color of the layer to dominate on the final image. In other words, if we make I bias very negative, a weaker degree of gray will be enough to make the corresponding color appear on the resulting image. This explains the different values adopted for I. The result was further processed by the following program:

```
(main map.gif
(w→red, map.gif→blue || map.gif→red)
(w→green, map.gif→green ^ map.gif→blue)
(output (out,w))).
```

This program changes the white color into red, allowing us to obtain the result that we were looking for [see Fig. 12(c)].

VII. Conclusion

When low-level hardware simulations of CNN's are very costly for exploring new applications, the use of a behavioral simulator becomes indispensable. The system hereby presented allows to explore new color image processing applications in short turn around times. Processing with CNN's is attractive because the continuous transition from state to state shows the evolution of the image to its final appearance. Although the systematic development of programming templates is still an active area of research, the procedure to process an image using only two templates is appealing for its simplicity. The work hereby introduced advanced the state of the art into processing of color images. It was demonstrated that by collecting results from the cell's state rather than from its soft limited output, it is possible to obtain a full gamut of color tones. Two color mapping schemes were introduced that effectively assign states to distinct color hues. The error produced by these schemes is minimum. Therefore, they are deemed to be suitable for CNN color simulations.

Finally, from the examples presented in the last section, one can see the unquestionable potential of CNN in image processing applications.

Acknowledgment

The authors thank M. Basso for developing the templates to process the weather map image and to Prof. T. Roska for early discussions on this work.

References

[1] L. O. Chua and T. Roska, "The CNN paradigm," *IEEE Trans. Circuits Syst. I*, vol. 40, pp. 147–156, Mar. 1993.
[2] T. Matsumoto, T. Yokohama, H. Suzuki, and R. Furukawa, "Several image processing examples by CNN," in *Proc. IEEE Int. Wkshp. Cellular Neural Networks Applicat.*, 1990, pp. 100–111.
[3] T. Matsumoto, L. O. Chua, and H. Suzuki, "CNN cloning template: Shadow detector," *IEEE Trans. Circuits Syst.*, vol. 37, pp. 1070–1073, Aug. 1990.
[4] ——, "CNN cloning template: Connected component detector," *IEEE Trans. Circuits Syst.*, vol. 37, pp. 633–635, May 1990.
[5] T. Matsumoto, L. O. Chua, and T. Yokohama, "Image thinning with a cellular neural network," *IEEE Trans. Circuits Syst.*, vol. 37, pp. 633–635, May 1990.
[6] K. R. Crounse, T. Roska, and L. O. Chua, "Image halftoning with cellular neural networks," *IEEE Trans. Circuits Syst.*, vol. 40, pp. 267–283, Apr. 1993.
[7] T. Roska, A. Zarándy, and L. O. Chua, "Color image processing using multilayer CNN structure," in *Circuit Theory and Design '93*, H. Didiev, Ed. New York: Elsevier, 1993.
[8] L. O. Chua and B. E. Shi, "Multiple layer cellular neural networks: A tutorial," in *Algorithms and Parallel VLSI Architectures*, E. F. Deprettere and A. van der Veen, Eds., vol. A: Tutorials. New York: Elsevier, 1991, pp. 137–168.
[9] J. A. Nossek and T. Roska, Guest Eds., Special issue on cellular neural networks, in *IEEE Trans. Circuits Syst.*, vol. 40, 1993.
[10] C. C. Lee and J. Pineda de Gyvez, "Single-layer CNN simulator," in *Proc. IEEE Int. Symp. Circuits Syst.*, 1994.
[11] ——, "Time-multiplexing CNN simulator," in *Proc. IEEE Int. Symp. Circuits Syst.*, 1994.
[12] T. Roska and L. Chua, "The CNN universal machine: An analogic array computer," *IEEE Trans. Circuits Syst. II*, vol. 40, pp. 163–173, Mar. 1993.
[13] L. O. Chua and L. Yang, "Cellular neural networks: Theory," *IEEE Trans. Circuits Syst.*, vol. CAS-35, pp. 1257–1272, 1988.
[14] ——, "Cellular neural networks: Applications," *IEEE Trans. Circuits Syst.*, vol. CAS-35, pp. 1273–1290, 1988.
[15] K. Slot, "Determination of cellular neural network parameters for feature detection of two dimensional images," in *Proc. IEEE Int. Wkshp. Cellular Neural Networks Applicat.*, 1990, pp. 82–91.
[16] L. O. Chua and P. Thiran, "An analytic method for designing simple cellular neural networks," *IEEE Trans. Circuits Syst.*, vol. 38, pp. 1332–1341, 1991.
[17] W. K. Pratt, *Digital Image Processing*, 2nd ed. New York: Wiley, 1991.
[18] R. C. Gonzales and R. E. Woods, *Digital Image Processing*. Reading, MA: Addison-Wesley, 1992.
[19] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes. The Art of Scientific Computing*. Cambridge, U.K.: Cambridge Univ. Press, 1986.
[20] D. Koblas, "XPaint," public domain software.
[21] L. O. Chua and T. Roska, "The CNN universal machine, Part 1: The architecture," in *Proc. Int. Wkshp. Cellular Neural Networks Applicat. (CNNA)*, 1992, pp. 1–10.
[22] F. Zou, S. Schwarz, and J. A. Nossek, "Cellular neural-network design using a learning algorithm," in *Proc. IEEE Int. Wkshp. Cellular Neural Networks Applicat.*, 1990, pp. 73–81.
[23] P. Szolgay, I. Kispal, and T. Kozek, "An experimental system for optical detection of layout errors of printed circuit boards using learned CNN templates," in *Proc. 2nd IEEE Int. Wkshp. Cellular Neural Networks Applicat.*, Munich, Germany, Oct. 1992, pp. 203–209.
[24] T. Matsumoto, L. O. Chua, and R. Furokawa, "CNN cloning template: Hole filler," *IEEE Trans. Circuits Syst.*, vol. 37, pp. 635–638, May 1990.
[25] "CNN analogic dual software library," Comput. Automat. Inst., Hungarian Academy Sci., Internal Rep. DNS-1-1993, Jan. 1993.

Chi-Chien (Brian) Lee received the B.S. degree in electrical engineering from Texas A&M University, College Station, in 1991. He is currently working toward the M.S.E.E. degree there, with a research concentration on software simulation methods for cellular neural networks CNN's and color image processing with CNN's.

He was a Teaching Assistant in the Electrical Engineering Department of Texas A&M from August 1991 to December 1991. He is also coauthor of two papers presented in ISCAS'94 on CNN software simulation. He is currently working at Crystal Semiconductor in Austin, TX, as a Mixed-Signal Product/Test Engineer.

Mr. Lee is a member of Eta Kappa Nu and Tau Beta Pi National Honor Societies.

Jose Pineda de Gyvez (S'88–M'90) received the bachelor's degree in electronic systems engineering from the Technological Institute of Monterrey, Mexico, with a major in computer engineering, the M.Sc. degree from the National Institute of Astrophysics Optics, and Electronics, Tonanzintla, Mexico, and the Ph.D. degree from the Eindhoven University of Technology, the Netherlands, in 1982, 1984, and 1991, respectively.

He was a Junior Scientist with the Foundation for Fundamental Research on Matter, The Netherlands, from August 1986 to February 1991, working on CAD for yield, defect, and fault modeling. He is currently an Assistant Professor in the Department of Electrical Engineering and also holds a joint Faculty appointment with the Department of Computer Science, both at Texas A&M University, College Station. He is the author of *Integrated Circuit Defect Sensitivity: Theory and Computational Models* (Boston, MA: Kluwer, 1991). His research interests include analog signal processing, neural networks, and IC manufacturability.

Dr. Pineda is a member of the ACM. He is currently Associate Editor for technology of IEEE TRANSACTIONS ON SEMICONDUCTOR MANUFACTURING and Associate Editor for cellular neural networks of IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS I.