XCNN: A SOFTWARE PACKAGE FOR COLOR IMAGE PROCESSING

Jose Pineda de Gyvez¹

Department of Electrical Engineering
Texas A&M University, College Station, TX, 77843

Abstract

This paper presents a software prototype capable of performing image processing applications using Cellular Neural Networks (CNN). The software is based on a CNN multi-layer 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. Additionally, a post processor capable of performing pixelwise logical operations among color layers was developed to enhance the results obtained from CNN.

1. 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 CNN paradigm was created only for 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 multi-layer CNN[8]. Simulation strategies based on CNN multi-layer 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> 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.

1. This work was supported by the Office of Naval Research under contract grant number N00014-91-1-0516.

The basic structure of the simulator presented in this paper is based on a high performance software capable of efficiently dealing with large images in the order of 10^5 pixels [9, 10]. 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 the Graphics Interface Format (GIF) format as standard input.

2. COLOR PROCESSING

The RGB color model used in the simulator is an *additive primary system* that describes a color in terms of the percentage of red, green, and blue in the color[5]. 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. $\langle 255, 255, 255 \rangle$ in RGB values. Conversely, combining 0% red, 0% green, and 0% blue creates black, $\langle 0, 0, 0 \rangle$ in RGB values. The model chosen for color image processing with CNN is the RGB model. 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 $\mathcal{L}_R$, $\mathcal{L}_G$, $\mathcal{L}_B$. Thus, a simulation approach is to have the triplet $\langle RGB \rangle$ 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 $\mathcal{L}_R$, and averaging in $\mathcal{L}_G$, simultaneously.

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

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

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

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. $\underline{A}$ and $\underline{B}$ are block triangular matrices and $\underline{I}$, $\underline{x}$, $\underline{y}$ are vectors as follows:

$$\underline{A} = \begin{bmatrix} A_r & 0 & 0 \\ A_{rg} & A_g & 0 \\ A_{rb} & A_{gb} & A_b \end{bmatrix} \quad \underline{B} = \begin{bmatrix} B_r & 0 & 0 \\ B_{gb} & B_g & 0 \\ B_{rb} & B_{gb} & B_b \end{bmatrix} \quad (2a)$$

$$\underline{x} = \begin{bmatrix} x_{rij} \\ x_{gij} \\ x_{bij} \end{bmatrix} \quad \underline{y} = \begin{bmatrix} y_{rij} \\ y_{gij} \\ y_{bij} \end{bmatrix} \quad \underline{u} = \begin{bmatrix} u_{rij} \\ u_{gij} \\ u_{bij} \end{bmatrix} \quad \underline{I} = \begin{bmatrix} I_{rij} \\ I_{gij} \\ I_{bij} \end{bmatrix} \quad (2b)$$

where subindexes r, g, b have been used to refer to color layers $\mathcal{L}_R$, $\mathcal{L}_G$, $\mathcal{L}_B$, respectively.

Recall now from equation (2) that the CNN's output function is binary. 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, the color information is taken from the cell's state rather than from the output itself. Notice however that the cell's state, x , is not bounded, and while with fully saturated colors there is a straight mapping from CNN output values to color intensities, e.g. $C : \{-1,1\} \rightarrow \{0, 255\}$, the problem here is more complex as the state x can take any value. In other words, we need to find a function capable of mapping all real numbers to the closed interval $[0,255]$, e.g. $C : \mathbb{R} \rightarrow \{0,1,2,\dots, 255\}$. 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. The general mapping is as follows

$$z = \frac{J}{2} + \frac{J}{2g} \log_2 \left\{ (G - 1) \left[\frac{(x - 1)}{(x_u - 1)} \right] \right\} \quad \text{for } x > 1 \quad (3a)$$

$$z = \frac{J}{2} - \frac{J}{2g} \log_2 \left\{ (G - 1) \left[\frac{(x + 1)}{(x_u + 1)} \right] \right\} \quad \text{for } x < -1 \quad (3b)$$

where J is the absolute maximum color value, i.e. $J = 255$. Notice that this value is split at half the color range. This is an arbitrary cut-off which has given us good visual perceptive results. For the continuous mode mapping, the bilinear transformation employed in analog to digital filter transformations was applied. 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 and 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 and 255 . The general form of the bilinear transformation is given by

$$z = \frac{J}{2} - \left\{ \frac{J}{2} \left[\frac{\bar{x} + 1}{1 - \bar{x}} \right] \right\} \quad \text{for } x > 1 \quad (4a)$$

$$z = \frac{J}{2} - \left\{ \frac{J}{2} \left[\frac{x + 1}{1 - x} \right] \right\} \quad \text{for } x < -1 \quad (4b)$$

where $\bar{x}$ is taken as the negative of the current state value.

3. A CNN POST-PROCESSOR

Doing image processing with CNN may not always yield the desired results and post-processing becomes then necessary. The CNN post-processor consists of a compiler capable of handling logical pixelwise operations among distinct color layers. The added capability allows to create new processed images with say, one layer processed by CNN and the remaining layers logically manipulated between CNN results and the original image. The syntax of the post-processing language using a Backus Naur Form notation is shown next; keywords and variables are identified as boldface and italic words, respectively.

```
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) ")" ")" .

```

All the files to be processed by this CNN pot-processor must be specified at the beginning of the program as indicated in statement 1. 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.

4. COLOR IMAGE PROCESSING USING CNN

This example deals with the X-rays image of a chest cage (148,370 pixels) displayed in Fig. 1a. 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 through the *B* template. The resulting image was further processed by a "filler template". The template was applied only to the red layer; the other two remaining layers were processed with a "unity" template which basically filters the image without modifications. These templates are as follows

$$\text{Peeler: } A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 & 0 \\ 3 & 3 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad I = -I \quad (5a)$$

$$\text{Filler: } A = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad I = -I \quad (5b)$$

Basically, the function of the *A* template is to feed the pixel values of the neighboring pixels into the current pixel. A negative bias is used to avoid having an excessively dark image. The resulting

image was post-processed to achieve an optimal color manipulation. The post-processing 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. *i)* The red layer of both edge detection operations is ORed to obtain soft and hard tone contours, *ii)* the green layer of the original image is ORed with the green layer of the edge detection obtained using the quantized color mode, and *iii)* the blue layer is left intact. The result of this post processing is shown in Fig. 1b. 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.

Fig. 1. Processing of chest cage X-Rays image. (a) Original. (b) After processing with XCNN

The second example deals with color contrasting. Let us bring your attention to the top region of Fig. 2a (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[21]

$$A = \begin{bmatrix} 0.01 & -0.075 & 0.01 \\ -0.075 & 1.28 & -0.075 \\ 0.01 & -0.075 & 0.01 \end{bmatrix} \quad B = \begin{bmatrix} -0.04 & -0.13 & -0.04 \\ -0.12 & 0.71 & -0.13 \\ -0.04 & -0.13 & -0.04 \end{bmatrix} \quad I = -0.365 \quad (6)$$

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 3 iterations. The resulting processed image with the clouds uncovered is shown in Fig. 2b. Fig. 2c 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 the results.

Fig. 2. (a) Original image. (b) Contrasting operation. (c) Edge Detection

REFERENCES

- [1] L.O. Chua and T. Roska, "The CNN Paradigm," *IEEE Trans. on Circuits and Systems -I: Fundamental Theory and Applications*, vol 40, no. 3, pp. 147-156, March 1993.
- [2] T. Matsumoto, T. Yokohama, H. Suzuki, and R. Furukawa, "Several image processing examples by CNN," *Proc. IEEE Int. Workshop on Cellular Neural Networks and Their Applications*, pp.100-11, 1990.
- [3] T. Matsumoto, L.O. Chua, and H. Suzuki, "CNN clonning template: Shadow detector," *IEEE Trans. Circuits Syst.*, vol 37, pp. 1070-1073, Aug. 1990.
- [4] T. Matsumoto, L.O. Chua, and H. Suzuki, "CNN clonning 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 Thining with a Cellular Neural Network," *IEEE Trans. on Cricuits 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. Ciruits & Syst.*, vol. 40, pp. 267-283, April 1993.
- [7] T. Roska, A. Zarándy, L.O. Chua, "Color Image Processing using Multi-layer CNN Structure," *H Didiev (ed), Circuit Theory and Design 93*, Elsevier, Amsterdam, 1993.
- [8] L.O. Chua and B.E. Shi, "Multiple Layer Cellular Neural Networks: A tutorial," *Algorithms and Parallel VLSI Architectures*, E.F. Depretere and A. van der Veen, Eds., vol. A: Tutorials, New York: Elsevier, 1991, pp. 137-168.
- [9] C.C. Lee and J. Pineda de Gyvez, "Single-Layer CNN Simulator," *Proc. IEEE Int. Symposium on Circuits and Syst.*, 1994
- [10] C.C. Lee and J. Pineda de Gyvez, "Time-Multiplexing CNN Simulator", *Proc. IEEE Int. Symposium on Circuits and Syst.*, 1994