ISCAS 2000 - IEEE International Symposium on Circuits and Systems, May 28-31, 2000, Geneva, Switzerland

Implementation of Non-Linear Templates using a Decomposition Technique by a 0.5μm CMOS CNN Universal Chip.

G. Liñán, P. Foldesy, A. Rodríguez-Vázquez, S. Espejo and R. Domínguez-Castro.
Instituto de Microelectrónica de Sevilla – CNM-CSIC
Edificio CICA-CNM, C/Tarfia s/n, 41012- Sevilla, SPAIN
Phone: +34 95 4239923, Fax: +34 95 4231832, E-mail: linan@imse.cnm.es

ABSTRACT[†]

This paper demonstrates the processing capabilities of a recently designed Analog Programmable Array Processor [1]. This new prototype, that is called CNNUC3, follows the Cellular Neural Network Universal Machine computing paradigm [2], [3], [4]. Due to its very advanced features and algorithmic capabilities, this chip has been demonstrated to be able to perform not only linear templates executions, but also to be very adequate for the implementation of non-linear templates by using a decomposition method. This paper focus on the application examples of the execution of non-linear templates with the CNNUC3 prototype. A brief description of the theoretical background is also presented in the paper.

1. INTRODUCTION.

Cellular Neural Networks (CNNs) [2] exhibits outstanding image processing capabilities. When considering the CNN paradigm, linear and nonlinear operations (so called templates) can be distinguished. The linear operations are mainly linear convolutions among the pixel values, regardless the value of the pixel that is being processed. On the other hand, the nonlinear templates present the property of changing or adapting the strength of the connections between different cells (weights) according to the current value of the pixels under operation.

These nonlinear functions play an important role in image processing. However, the nowadays available CNN implementations are not capable to implement such operations because of the hardware difficulty of implementing, and moreover make them programmable, non-linearities. To solve this problem some algorithmic methods have been developed [5]. These algorithms use simple nonlinear functions and extensions of the original CNN paradigm. Fortunately, these extensions are defined in the CNN Universal Machine architecture [3], [4], which comprises the gray-scale (or analog) and binary (or logic) operations with distributed internal memories.

The CNNUC3 prototype [1] is, by far, the most complex CNN implementation reported up to now. This is the first high-density CNN chip that can process and provide gray-scale images also containing many advanced features pointing towards the CNNUM. Among these extensions we could emphasize:

- The algorithmic capability of the chip is enough to run algorithms with dozens of operations without external code or data movement.
- It can store four gray-scale and four binary images.
- It can sum or subtract gray-scale images.
- It has the capability of selecting which cells are going to be processed (so called freezing map).
- It is possible to combine two binary images by any logic operation (such as logic "and", "or", to that purpose, it contains a fully programmable two input digital device within each cell).

The paper is organized as follows; Section 2 establishes a theoretical background about the technique of non-linear templates decomposition. Section 3 describes some applications examples. Some additional comments are provided in Section 4. Finally, the conclusions are presented in Section 5.

2. DECOMPOSITION OF NON-LINEAR TEMPLATES.

Implementing a non-linear template by decomposing it into the execution of several linear ones is not a new problem for templates engineers. In this section we will briefly describe the method reported in [5] in order to accomplish this transformation.

Therefore, we will deal with the decomposition of 3×3 templates where only the **B** term is a non-linear function. Furthermore, we will assume that the non-linearities appearing on the feedforward term are piecewise linear functions and that the input image is time invariant.

With these assumptions, the dynamic evolution of a cell (considering the FSR model [6]) is given by:

$$\tau\frac{dx_{ij}(t)}{dt} = -g[x_{ij}(t)] + \sum_{C(k,l) \in S_r(i,j)} A(i,j;k,l)x_{kl}(t) + \\ + \sum_{C(k,l) \in S_r(i,j)} B(i,j;k,l) \bullet u_{kl} + z \qquad (1)$$

†. This work has been partially funded by ONR-NICOP N68171-98-C-9004 and DICTAM IST-1999-19007.

$$g[x_{ij}(t)] = \begin{cases} m_L & , x_{ij}(t) < -1 \\ 0 & , |x_{ij}(t)| < 1 \\ m_R & , x_{ij}(t) > 1 \end{cases} \qquad \begin{matrix} m_L \to -\infty \\ m_R \to \infty \end{matrix} \tag{2}$$

The problem is how to substitute the non-linearities associated to the **B** term by using a sequence of linear templates.

Let us suppose that the non-linear piecewise function can be expressed as:

$$\Psi(\xi) = \Psi(\alpha \cdot u_{ij} + \beta \cdot u_{kl}) \tag{3}$$

where α and β are real numbers, and that the linear regions are defined by a set of m breaking points $\{\xi_1, \xi_2, \dots, \xi_m\}$. In that case, that is also the most common in practice, the non-linear template can be decomposed into a sequence of linear template executions. The algorithm that is exhaustively described and examined in [5], runs as follows:

- The process starts by selecting the first linear region of the non-linear function. Let us call R_1 this region that is defined by the breaking points ξ_1, ξ_2.
- The next step is to select which are the cells belonging to that region. This calculation is realized by two templates executions and a logic operation (all of them are done on-chip). With the first template, the so called threshold template, we drive to black all those cells having $\xi > \xi_1$, while with the second one, the so called inverse threshold, we drive to black all those cells having $\xi < \xi_2$. Finally a logic AND operation of both results will select those pixels where $\xi_1 < \xi < \xi_2$ [‡].

  Equations (4), (5), show the threshold and the inverse threshold template[††].

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} \beta & 0 & 0 \\ 0 & \alpha & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad z = -\xi_1 \tag{4}$$

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} -\beta & 0 & 0 \\ 0 & -\alpha & 0 \\ 0 & 0 & 0 \end{bmatrix}, \qquad z = \xi_2 \tag{5}$$

- The non-selected cells are "frozen", by using the freezing mask provided by the chip, while in the selected ones the corresponding contribution to the state equation is evaluated and stored as a "bias map" that will be updated (or not) in the next iteration by adding the new result to the one that was previously stored. The updating law for the state variables of the cells that are selected must be given by the equation of a straight line (due to the fact that $\Psi(\xi)$ is linear between each two breaking points) crossing the points ξ_1 and ξ_2. All the points belonging to this line satisfy:

$$\frac{\xi - \xi_1}{\xi_2 - \xi_1} = \frac{\Psi(\xi) - \Psi(\xi_1)}{\Psi(\xi_2) - \Psi(\xi_1)} \tag{6}$$

And from the CNN theory, it can be demonstrated that this relationship is obtained if the following template is executed[‡‡]:

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} k \cdot \beta & 0 & 0 \\ 0 & k \cdot \alpha & 0 \\ 0 & 0 & 0 \end{bmatrix}$$
$$z = \Psi(\xi_1) - k \cdot \xi_1 \tag{7}$$

where,

$$k = \frac{\Psi(\xi_2) - \Psi(\xi_1)}{\xi_2 - \xi_1} \tag{8}$$

- The process continues for the next linear region.
- Finally, a template execution is needed. In this template the feedback term is the same as in the original one defined in (1), the feedforward term is set to zero (modified B template), since it has been already calculated, and the offset term is the addition of the original one z, and the *"bias map"* that is stored in some memory on the cell.

3. APPLICATION EXAMPLES.

3.1 Absolute Value Calculation.

In this subsection we consider only pixel-wise transformations, or with other words, B templates with the size of $1x1$.

As a consequence of missing neighbor connections the decomposition method can be simplified, avoiding the accumulation of the partial results. Moreover, the selection of cells belonging a given interval is done by the two threshold templates, which also contain only central elements (where $\alpha = 1$, $\beta = 0$ and $\xi_1 = 0$):

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & \alpha & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad z = -\xi_1 \tag{9}$$

As an example we show how the absolute value can be calculated. The used operation and template is shown in Fig..

$$A = [1] \quad B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & \Psi(\xi) & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad z = 0$$

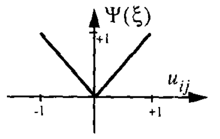

Fig. 1: The absolute value calculation template.

Since there are two intervals, the positive and negative valued cells, there are two cell maps. The first one contains black pixels at the cell positions where the input image contained negative values and the second is the opposite of it. As a special case, the first

‡. Keep in mind that $\xi = \alpha \cdot u_{ij} + \beta \cdot u_{kl}$ and the subindex kl denotes the cell' neighbors.

††. These are the FSR version of the templates. In order to get the original Chua-Yang template increase by one the self-feedback term.

‡‡. The position of the β coefficient must be rotated in order to perform this operation for each of the neighbors of the cell appearing as a non-linear connection on the original B template. Therefore, each linear region could require up to 16 templates and 8 logic operations to be selected, 8 templates to update the state variable, and 8 templates to perform the addition of the results, that is 32 templates and 8 logic operations.

transformation is equal to inversion and the second one practically can be avoided (since it lets the cells unchanged at their original values). Fig.. shows the result of the execution of the absolute value calculation.

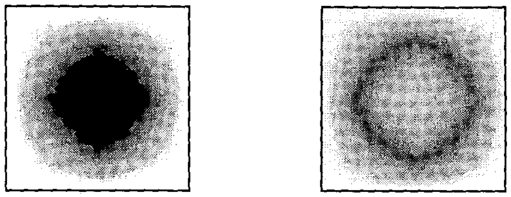

Fig. 2: The absolute value calculation.

3.2 Gradient calculation

The second example is the calculation of the gradient and the thresholded gradient.

The gradient template is defined as follows:

$$A = \begin{bmatrix} 1 \end{bmatrix}$$

$$B = \begin{bmatrix} \Psi(\xi) & \Psi(\xi) & \Psi(\xi) \\ \Psi(\xi) & 0 & \Psi(\xi) \\ \Psi(\xi) & \Psi(\xi) & \Psi(\xi) \end{bmatrix}$$

$$z = 0$$

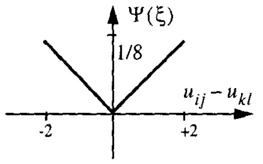

Fig. 3: The gradient template.

The template contains eight neighboring connections that can belong to two intervals. After the usage of the decomposition method the total number of linear template executions and threshold functions is 32 .

The thresholded gradient operation differs from the gradient calculations in the values of the modified **B** template.

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad z = z_{threshold} \qquad (10)$$

Execution examples can be seen in Fig.. and in Fig..

3.3 Contour Detection on Gray-Scale Images.

The third example is the contour detection. The operation is defined in such a way that the output contains black pixel at the cell

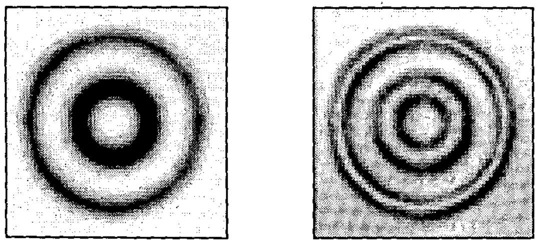

Fig. 4: The gradient calculation.

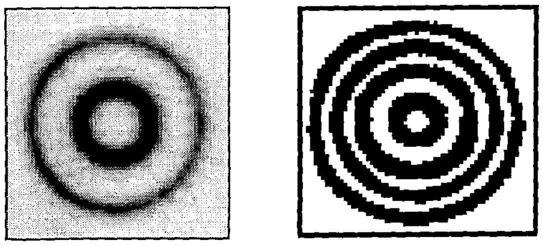

Fig. 5: The thresholded gradient extraction.

position where the input value of the cell is larger than some of the neighbors by a certain amount (0.1 in the case of Fig.).

$$A = \begin{bmatrix} 2 \end{bmatrix}$$

$$B = \begin{bmatrix} \Psi(\xi) & \Psi(\xi) & \Psi(\xi) \\ \Psi(\xi) & 0 & \Psi(\xi) \\ \Psi(\xi) & \Psi(\xi) & \Psi(\xi) \end{bmatrix}$$

$$z = -0.5$$

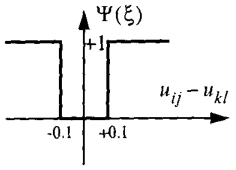

Fig. 6: The contour Detection Template.

Both the number of used mask generating templates and transformation templates are 16 †††. The result of the execution of this sequence to a gray scale image can be observed in Fig..

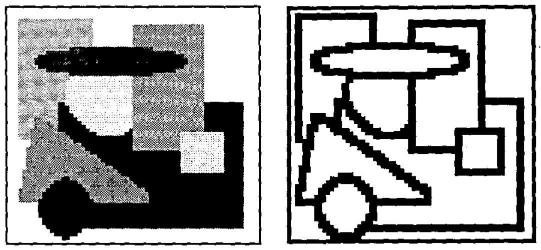

Fig. 7: Contour Detection on Gray-Scale Images.

3.4 Local Maxima

This example shows how the local (3×3) maxima can be extracted. The cell's output is black (or contains a local maxima) if the cell's input value is larger by certain amount (0.05 in the case of the template in Fig.) than any of the neighbors.

The decomposition is similar to the previous one, but in this case all of the partial results should provide a positive detection, while the contour operation required only one positive detection. The decomposed sequence contains 8 templates. An example of the application of this template can be seen in Fig..

†††. See that the number of required templates is not 64 as it should correspond to the case of having 8 non-linear connections. This is explained by the fact that the linear regions have an infinite or zero slope, and so, the linear transformation defined by (7) is not needed.

$$A = \begin{bmatrix} 2 \end{bmatrix}$$

$$B = \begin{bmatrix} \Psi(\xi) & \Psi(\xi) & \Psi(\xi) \\ \Psi(\xi) & 0 & \Psi(\xi) \\ \Psi(\xi) & \Psi(\xi) & \Psi(\xi) \end{bmatrix}$$

$$z = -7.5$$

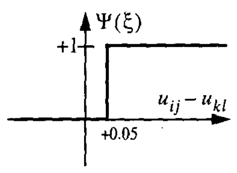

Fig. 8: The Local Maxima Template

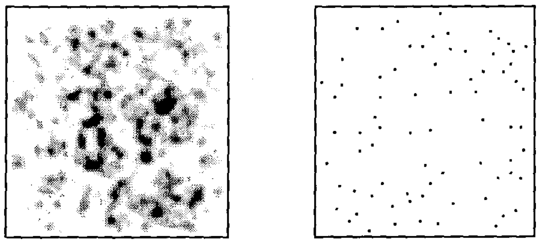

Fig. 9: The local maxima detection.

4. ADDITIONAL COMMENTS.

In this section we mention some additional ideas about the decomposition, which reduce the number of the required operations. This reduction arises from some special functions that are available in the CNNUC3 chip.

- The first example shows that when the number of intervals is only two, the "freezing" masks are the opposite of each other. This implies that the calculation of the second mask by template execution can be replaced by a logic operation.
- The third and fourth examples demonstrate that there are special cases when the general method can be modified in order to get a more efficient decomposition. Specially, when the partial results contains only black or white pixels.

  In these cases, the generated interval maps contain all the information about the partial results. That means that the linear transformation (the third step of the algorithm in Section 2) is not needed. Moreover, when the final result is the logic sum (operation OR) or logic product (operation AND) of the partial outputs, the final result can be accumulated by the Local Logic Unit (LLU) in a Local Logic Memory (LLM) instead of by using the gray-scale accumulation process in an analog memory.

4.1 Processing, Precision, and Time

Since we use an VLSI analog implementation, precision and processing time are important issues that should be mentioned.

The global precision of the chip is slightly below 8 bits, that refers to the spatial uniformity. On the other hand, the nonlinear-to-linear transformation of a piecewise function containing about 8-10 breaking points is possible. Furthermore, a non-linear function not belonging to the piecewise class, could also be implemented if there exist a good enough piecewise approximation (containing up to 10 breaking points).

The processing time of a single template execution and a logic operation are $20\mu s$ (including internal calibrating phases and the settling time for changing the template coefficients) and $1\mu s$ respectively.

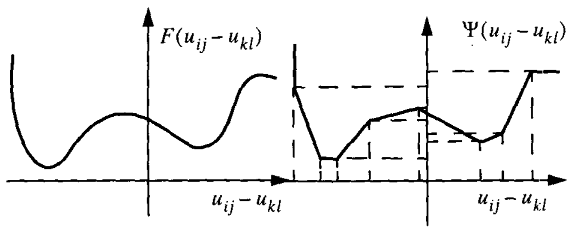

Fig. 10: Piecewise approximation of a generic function

5. CONCLUSIONS.

The executions of non-linear templates defines an important application area in the field of image processing. However, previous VLSI CNNs implementations did not provide to the template engineers sufficiently accurate and versatile features to map the nonlinear-to-linear existing algorithms. We have presented experimental evidences in this paper about how a wide set of non-linear templates can be executed with a reasonable accuracy with a recently designed CNN prototype, the so called CNNUC3. We have also briefly outlined a general decomposition method for implementing non-linear-to-linear template transformations.

6. REFERENCES.

[1] G. Liñán, P. Foldesy, S. Espejo, R. Domínguez-Castro and A. Rodríguez-Vázquez. " A 0.5mm CMOS 10^6 Transistors Analog Programmable Array Processor for Real-Time Image Processing", *Proc. of the 25th European Solid-State Circuits Conference*, pp. 358-36, Duisburg-Germany, Sept. 1999.

[2] L.O. Chua and L. Yang. "Cellular Neural Networks: Theory", IEEE Trans. Circuits and Systems, vol. 35, pp. 1257-1272, Oct. 1988.

[3] T. Roska and L.O. Chua. "The CNN Universal Machine: An Analogic Array Computer", IEEE Trans. Circuits and Systems II, Vol. 40, pp 163-173, March 1993.

[4] L.O. Chua and T. Roska. "The CNN Paradigm", *IEEE Trans. Circuits and Systems I*, vol.40, pp.147-156, March 1996.

[5] L. Kek and A. Zarandy. "Implementation of Large Neighborhood Non-Linear Templates on the CNN Universal Machine". *International Journal of Circuit Theory and Applications*, vol.26, No. 6, pp. 551-566, 1998.

[6] S. Espejo, R. Carmona, R. Domínguez-Castro and A. Rodríguez-Vázquez: "A VLSI-Oriented Continuous-Time CNN Model". *International Journal of Circuit Theory and Applications*. Vol 24, No. 3, pp 341-356, May-June 1996.