Large-image CNN hardware processing using a time multiplexing scheme

Citation for published version (APA):

Pineda de Gyvez, J., Wang, L., & Sanchez-Sinencio, E. (1996). Large-image CNN hardware processing using a time multiplexing scheme. In *Proceedings of the Fourth IEEE International Workshop on Cellular Neural Networks and their Applications, CNNA-96, 24-26 June 1996, Sevilla, Spain* (pp. 405-410). Institute of Electrical and Electronics Engineers. <https://doi.org/10.1109/CNNA.1996.566608>

DOI:

[10.1109/CNNA.1996.566608](https://doi.org/10.1109/CNNA.1996.566608)

Document status and date:

Published: 01/01/1996

Document Version:

Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

- A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
- The final author version and the galley proof are versions of the publication after peer review.
- The final published version features the final layout of the paper including the volume, issue and page numbers.

[Link to publication](#)

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy

If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Large-Image CNN Hardware Processing using a Time Multiplexing Scheme

Jose Pineda de Gyvez¹, Lei Wang and E. Sanchez-Sinencio

Department of Electrical Engineering

Texas A&M University

College Station, TX, 77843

Email: gyvez@pineda.tamu.edu Phone: +1-(409)-8457477

ABSTRACT: *The state of the art work in Cellular Neural Networks (CNN) has concentrated on VLSI implementations without really addressing the "systems level". While efficient implementations have been reported, no reports have been presented on the use of these implementations for processing large complex images. The work hereby presented introduces a strategy to process large images using small CNN arrays. The approach, time-multiplexing, is prompted by the need to simulate hardware models and test hardware implementations of CNN. For practical size applications, due to hardware limitations, it is impossible to have a one-on-one mapping between the CNN hardware processors and all the pixels in the image involved. This paper presents a practical solution by processing the input image block by block, with the number of pixels in a block being the same as the number of CNN processors in the hardware. Image processing results obtained from an actual IC test-chip prototype using this scheme are presented.*

1. Introduction

While software prototypes prove the potential of CNN [1,2], a great deal of research has been advocated to hardware implementations which can be used for on-line applications in real-time[3-9]. Unfortunately, even though these implementations are extremely efficient the problem of how to use them for processing large images has not been properly addressed. For practical image size applications, due to current state of the art technological limitations, it is impossible to have a one-on-one mapping between the CNN hardware processors and all the pixels in the image involved. It is thus a key issue the proper use of these implementations in common-day situations. This paper presents a practical solution by processing the input image block by block, with the number of pixels in a block being the same as the number of CNN processors in the hardware.

2. Time-Multiplexing Hardware Simulation

Under this approach one can define a block of pixels (subimage) which will be processed by an equal number of CNN cells. Once convergence is achieved, a new subimage adjacent to the one just processed, is scheduled for further processing. This procedure is repeated until the whole image has been scanned using a lexicographical order, say, from left to right and from top to bottom. It is obvious that with this approach the processing of large images becomes feasible in spite of the finite number of CNN cells.

Even though the approach seems simple and appealing, an important observation is necessary: *The processed border pixels in each subimage may have incorrect values since they are processed without neighboring information.* Fortunately, the latency of CNNs is such that only local interactions, depending upon the neighborhood radius, are important. Hence, to cope with the previous problem, two sufficient conditions must be considered to ensure that each border cell properly interacts with its neighbors. These conditions are: 1) to have a *belt of pixels* from the original image around the subimage being processed, and 2) to have *pixel overlaps* between adjacent subimages. We will go into the details of these two constraints in the next subsection.

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

Fig.1. Time-Multiplexing boundary conditions. (a) Belt of inputs. (b) Overlapped Cells

2.1 Sufficient and Necessary Conditions for Time Multiplexing

As an example of processing error, consider the case of Fig. 2 in which a 4x4 CNN is processing one section of an image. Assume further that all the entries of the B template are ones. Then under the conditions depicted, cell $C(3,4)$ will present a 50% computation error if the pixels surrounding the border are not considered. This can be concluded by observing that the total weighted input to the state of cell $C(3,4)$, without considering border pixels, is 6 units instead of 3. It is possible to quantize the processing error of any border cell $C(i,j)$ within a given neighborhood radius. Let us compute independently the error due to the feedforward operator and then due to interactions among cells for two horizontally adjacent processing blocks. Assume that the neighborhood for a border cell is incomplete, e.g. we are missing the data coming from pixels out of the CNN array. Then, the absolute processing error of a border cell $C(p,q)$ due only to the effect of the B template is obtained by subtracting the erroneous state value from the error free states. The erroneous state value is characterized by the absence of external input signals to the cell. This yields the following result

$$\varepsilon_{ij}^B = \sum_{C(k,l) \in N(p,q)} B^*(p,q;k,l) \text{sign}(u_{k,l}) \quad (1)$$

where $B^*(p,q;k,l)$ are the missing entries from the B template due to the absence of input signals $u_{k,l}$ and $\text{sign}(\cdot)$ is the sign function. The latter function is used to represent the status of a pixel, e.g. *black* = 1 and *white* = -1. Notice that the error is both image and template dependent. In other words, the steady state of a border cell may converge to an incorrect value due to the absence of its neighbors weighted input. One can easily conclude that the error is canceled if the missing external inputs are provided to the border cells as depicted in Fig. 1a. Since typically, the array is "embedded" in the image during operation, this condition can easily be satisfied.

Fig. 2. Error quantification due to missing neighbor pixels using only the B template

Let us address now the interactions among cells. For this effect, we can compute the absolute error in a similar form. Disregarding for the moment the B template this error is

$$\varepsilon_{ij}^A = \sum_{C(k,l) \in N(p,q)} A^*(p,q;k,l) y_{k,l}(t) \quad (2)$$

where $A^*(p,q;k,l)$ are the missing entries from the A template due to the absence of weighted output signals $y_{k,l}(t)$. The problem in this situation is more involved because the output signals depend on the state of their corresponding cells. To

minimize the error an overlap of pixels between two adjacent blocks is proposed. In this form, the inner cells of the CNN array will always receive weighted processing information from the border cells.

The general time-multiplexing procedure consists in processing each image block until all CNN cells within the block converge. The block with converged cells will have state output variables y which are the values used for the final output image. Every time that a new subimage is processed, the physical CNN array is initialized to the initial conditions of the original image, or to black or to white as required by the template in use. In the overlapping procedure the outer overlapped cell's converged values are discarded since they were computed with incomplete neighboring information. Only the inner cell's converged values are kept as valid values. This implies that for a neighborhood radius of 1, an overlap of two pixel column/rows is needed to be able to ensure correct values for pixels assigned to the border cells. For instance, in Fig. 1b two pixel columns are overlapped. The converged values of the leftmost column are kept when processing $Block_k$ and the rightmost column values when processing $Block_{k+1}$.

Fig. 3. Pixel overlap; shadowed pixels represent the ones that are overlapped in two consecutive block scans. In an edge detection operation a black pixel is turned white only when it has two adjacent black neighbor pixels. (b) 1 pixel overlap; the left neighbor is missing which results in a wrong edge detection result. (c) 2 pixel overlaps; both adjacent neighbors are present.

To illustrate the previous statements, consider the image depicted in Fig. 3a and assume an edge detection operation. For this image processing operation the middle pixel is turned white only when its two neighbors are present. If only one pixel overlap is assumed the center pixel will never be white because when it is processed in both blocks it still misses the neighboring information, see Fig. 3b. This situation is fixed by overlapping two pixels. In particular, the three black pixels will be present when processing $Block_{k+1}$, see Fig. 3c. In our prototype the number of overlapping columns or rows between the adjacent blocks is defined by the user. An even number overlapping is recommended, since the converged cells in the overlapped region can be evenly divided by the two adjacent blocks. With the added overlapping feature, better neighboring interactions are achieved, but at the same time, an increase in computation time is inevitable. Consider an image of $M \times N$ pixels, a CNN array of $m \times n$ cells and a multiplexing scheme employing o overlaps. Then the total number of blocks (subimages) that need to be processed is

$$Blocks = \frac{M}{m-o} \times \frac{N}{n-o} \quad (3)$$

With the previous multiplexing scheme the image needs to be iterated several times over newly obtained states to allow the proper propagation of global effects. Multiple iterations are necessary to guarantee that all cells have converged to correct values taking into account all global effects. This can be inferred by considering a diagonal propagation of, say, a black pixel in a fully white image. Notice that without overlaps it is impossible to propagate global effects, see Fig. 4a, and that the propagation is achieved with at least one overlapped pixel. Fig. 4b depicts the case of three image iterations, each one acting over previously obtained results and using only one pixel overlap to propagate the data. Now, without loss of generality consider an $M \times M$ image, an $m \times m$ CNN array and a multiplexing scheme of o overlaps, with $o < m$. Then the minimum number of iterations needed to propagate a pixel along the diagonal of the image is given as

$$Iterations = \left\lceil \frac{M-o}{m-o} \right\rceil \quad (4)$$

where the symbol $\lceil \cdot \rceil$ is used for a ceiling function. Naturally, both the number of blocks and the number of iterations are directly proportional to the computational speed of the CNN system. By using eqs. (3) and (4) this computational speed can be estimated as follows

$$\text{Speed} = \text{Iterations} \times \text{Blocks} \times (\Phi + \tau) \quad (5)$$

where τ is the average convergence time of each block and Φ is the corresponding I/O transfer rate to handle the block sequencing operation.

Fig. 4. Propagation effects. (a) With no overlaps a pixel is propagated only until the boundary of the physical CNN array. (b) When overlaps occur, the propagation is achieved by processing the image over previously obtained results. In this case 3 iterations are necessary to propagate the pixel along the diagonal.

3. CNN Monolithic Prototype

The 3x3 CNN chip was fabricated with MOSIS n-well 2.0 μm process. The photograph of the die is shown in Fig. 5a, where all cells are arranged as a 3 by 3 array. The die area of the circuit is approximately 3.2 mm^2 . Unlike other implementations in which the output is observed at the hard-limiting block, the VLSI architecture we developed monitors the outputs from the state node. While previous implementations are mostly suitable for black and white applications because of the thresholded outputs, our approach is especially suitable for applications in color (gray) image processing due to the analog nature of the state node.

The CNN IC has shared input/output pins. Salient features of this implementation are full template programmability, a programmable integration time constant, and an external output at the state node. Fig. 5b shows a modular view of the CNN IC along with I/O signals.

- $b_{11}, b_{12}, \dots, b_{33}$ are the pins to set the analog values of template B
- $a_{11}, a_{12}, \dots, a_{33}$ are the pins to set the analog values of template A
- IO1, IO2, ... IO9 are the input-output pins of all nine cells. The pin of each cell is used to do the functions of setting the boundary conditions, initializing the state, and of providing external input values to the cell, as well as obtaining its state output.
- d1 and d2 are control signals to multiplex each input/output pin for different functions at different time periods
- V_{bias} is the offset bias voltage for the templates, and V_c is a tuning voltage of the active resistor
- 5V, -5V, 1V, -1V are power supplies for the circuit and for the activation function, respectively.

4. Experimental Results

The CNN chip is connected to a personal computer (PC) through an A/D and a D/A interfacing board. The operations of setting inputs and getting outputs from the CNN chip are multiplexed externally by 4-1 analog multiplexer chips

(ADG509A). External operations are synchronized with the multiplexing operations inside the chip. The type of A/D board was AT-MIO-16F, which has 12 A/D channels; the type of D/A board was AT-AO-6/10, which has 10 D/A channels. Both are products of National Instruments. The pin multiplexing control code is generated by a computer program and interfaced through the digital I/O port in the AT-MIO-16F board. Opamps were added as A/D output buffers to isolate the output node from the parasitic capacitance of the wires and the A/D board. The operating sequence is as follows:

Fig. 5 CNN Monolithic prototype. (a) Die photograph. (b) Pin and IC Floorplan

1. Initialize the A/D and D/A boards. Set the required template values by providing the corresponding analog voltages for the template values.
2. Set the boundary conditions of the CNN array, and the initial condition values of all cells
3. Map the pixel values (0 – 255) of the input image into CNN input voltage values (–1V to 1V), and send them to the chip
4. Extract the output values (–3V to 3V) of the state variables of all cells and map them to pixel values (0 – 255) of the output image.
5. Move to another position in the input image and repeat at step 2.

Fig. 6 shows the extracted results for an edge detection operation using this 3x3 CNN chip on a 256x256 image. Worth noting from the results is that the images have gray level colors.

Fig. 6. (a) Original image. (b) After edge detection with $A_{ii} > 1$. (c) After edge detection with $A_{ii} < 1$

5. Conclusions

This paper demonstrates the feasibility of processing large images using a time-multiplexing approach involving small CNN arrays. For practical image sizes, due to current state of the art technological limitations, it is impossible to have a

one-on-one mapping between the CNN cells and the pixels in the image involved. It was also shown that a state-node output approach is especially suitable for color image processing and applications involving continuous-time output signals.

References

- [1] C.C. Lee and J. Pineda de Gyvez, "Single-Layer CNN Simulator," *Proc. IEEE Int. Symposium on Circuits and Syst.*, pp. 217-220, 1994
- [2] C.C. Lee and J. Pineda de Gyvez, "Time-Multiplexing CNN Simulator", *Proc. IEEE Int. Symposium on Circuits and Syst.*, pp. 407-410, Dec. 1994
- [3] J.M. Cruz and L. Chua, "Design of High-Speed, High-Density CNNs in CMOS Technology," *Intl. Journal Circuit Theory and Applications*, vol. 20, pp.555-572, 1992.
- [4] A. Rodriguez-Vazquez, S. Espejo, R. Dominguez-Castro, J. L. Huertas, and E. Sanchez-Sinencio, "Current-Mode Techniques for the Implementation of Continuous and Discrete-Time Cellular Neural Networks," *IEEE Transactions on Circuits and Systems II*, vol. 40, no. 3, pp. 132-146, 1993.
- [5] J. E. Varrientos, E. Sanchez-Sinencio, and J. Ramirez-Angulo, "A Current-Mode Cellular Neural Network Implementation," *IEEE Transactions on Circuits and Systems II*, vol. 40, no. 3, pp. 147-155, 1993.
- [6] I. A. Baktir and M. A. Tan, "Analog CMOS Implementation of Cellular Neural Networks," *IEEE Transactions on Circuits and Systems*, vol. 40, no. 3, pp. 200-205, 1993.
- [7] G. F. Dalla Betta, S. Graffi, Zs. M. Kovacs, and G. Masetti, "CMOS Implementation of an Analogically Programmed Cellular Neural Network," *IEEE Transactions on Circuits and Systems-II*, vol. 40, no. 3, pp. 206-214, 1993.
- [8] J.A. Nossek and G. Seiler, "Cellular Neural Networks: Theory And Circuit Design," *Intl. Journal Circuit Theory and Applications*, vol. 20, pp.533-553, 1992.
- [9] Espejo S., et al, "A 1um CMOS cellular neural network machine," *Proc. IEEE Int. Workshop on Cellular Neural Networks Appl.*, CNNA-94 (Rome), 1994, pp. 91-96.