Cellular-Neural-Network Focal-Plane Processor as Pre-Processor for ConvNet Inference

Lionel C. Gontard
Computer Science and Eng. Dept.
University of Cádiz (Spain)
lionel.cervera@uca.es

Ricardo Carmona-Galán
Instituto de Microelectrónica de Sevilla
(CSIC-University of Seville, Spain)
rcarmona@imse-cnm.csic.es

Ángel Rodríguez-Vázquez
Instituto de Microelectrónica de Sevilla
(CSIC-University of Seville, Spain)
rcarmona@imse-cnm.csic.es

***Abstract*— Cellular Neural Networks (CNN [1]) can be embodied in the form of a focal-plane image processor. They represent a computing paradigm with evident advantages in terms of energy and resources. Their operation relies in the strong parallelization of the processing chain thanks to a distributed allocation of computing resources. In this way, image sensing and ultra-fast processing can be embedded in a single chip. This makes them good candidates for portable and/or distributed applications in fields like autonomous robots or smart cities. With the irruption of visual features learning through convolutional neural networks (ConvNets), several works attempt to implement this functionality within the CNN framework. In this paper we carry out some experiments on the implementation of ConvNets with CNN hardware in the form of a focal-plane image processor. It is shown that ultra-fast inference can be implemented, using as an example a LeNet-based ConvNet architecture.**

Keywords*— *Cellular Neural Networks, Focal-Plane Processors, Convolutional Neural Networks, Image Classification

I. INTRODUCTION

Deep learning in the form of Convolution Neural Networks (ConvNets) represents the state-of-the-art in computer vision applications. By exploiting feature learning, it has been particularly useful for solving complex tasks such as object detection and image segmentation and classification. After the training period is completed, the network can be deployed into the field for inference —processing data to infer a result. However, its use in real-time applications and embedded systems demands highly efficient and adapted edge-computing solutions. ConvNets have been implemented in GPUs, FPGAs and dedicated multicore processors called Vision Processing Units (VPUs) task [1]-[3].

One of the keys for energy efficiency is the adaptation of the processing architecture to the nature of the stimulus, in this case visual. Instead of following the traditional Von Neumann architecture, efficiency is fostered by embracing bioinspired organizational principles. In this approach, computing and memory resources are distributed across a vast number of relatively simple cells, called *neurons*. Each neuron communicates with hundreds or thousands of other neurons through so-called *synapses* [4][5]. This architecture displays many advantages over conventional sequential computers when dealing with sensory information. The distribution of resources eliminates a large fraction of data transfers across the system. This alleviates power consumption and improves temporal resolution. In this work, we explore the feasibility of implementing ultra-fast ConvNet inference for embedded real-time applications using a CNN-based Focal Plane Processor (FPP). This device is able to sense and process images at the sensor plane with the help of concurrent photodiodes and analog, logic and mixed-signal operators at every pixel.

The paper is distributed as follows. Section II briefly introduces the bioinspired architecture of the FPP, that is a CNN, and reviews how can it be applied for ConvNet inference. Section III describes the hardware platform used in this work, its features and limitations are described. Section IV briefly explains the operation of ConvNets. Section V, analyzes the implementation of the operators of ConvNet inference in the FPP. Finally, Section VI describes an algorithmic solution for implementing a LeNet ConvNet with the FPP, and its experimental evaluation in terms of processing speed.

II. RELATED WORK

The core of the FPP employed in this experiment is a CNN, a bioinspired computing paradigm invented 30 years ago in an effort to mimic the operation of the plexiform layers of the vertebrate retina. These networks (i) employ a grid based structure of similar processing elements, (ii) process information topologically, and (iii) employ analog hardware that can efficiently perform convolution operations with the so-called templates, that are the instruction primitives [6].

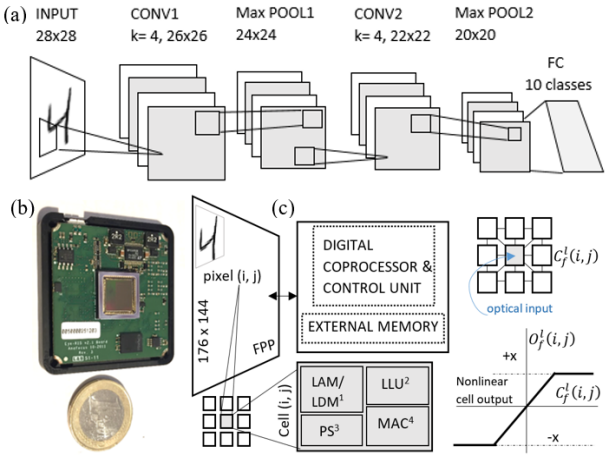

Fig. 1. (a) A version of LeNet implemented by Horváth et al. using 4 CNNs [9]. (b) CMOS chip used in this work featuring a focal plane processor of 176 × 144 pixels. (c) It contains a network of locally interconnected cells. Each cell has a photosensor (PS), local analog and digital memories (LAM and LDM), local logic unit (LLU), and a mixed-signal MAC. The output of the cell is a piecewise nonlinear function of the state.

[1] Along this paper, we will employ the abbreviation CNN to denote Cellular Neural Networks, as it was traditionally employed since 1988. Convolutional Neural Networks will be referred as ConvNets, instead of the nowadays common label CNN.

From the application point of view, the leap ahead of the CNN model was the invention of a Turing-complete CNN by Tamás Roska and Leon Chua in 1993, called the CNN Universal Machine (CNN-UM). Each cell of the CNN-UM has extended functionality and stored programmability. This provides a suitable platform for ultra-efficient image processing that has been extensively studied theoretically, using simulators and also implemented on a single chip using CMOS technology [7][8].

Recently, Horváth et al. proposed the use of a CNN for solving the MNIST digit recognition problem using a simplified version of the LeNet ConvNet shown in Fig. 1(a) [9]. They synthetized a hardware coprocessor containing four CNNs running in parallel, and they showed that the architecture is on par with state-of-the-art approaches when compared for classification accuracy as well as energy-delay per classification. The hardware implementation of the architecture requires however a number of CNNs working in parallel, as many as the maximum convolutional layer depth expected in the network. Particularly, they use 4 CNNs working in parallel to enable operations of convolutional layers with a maximum depth of 4. This is a practical limitation considering that ConvNets may use many more features maps in one single layer. Also, in their architecture, the time latency of image transfer to the net was not considered. In this work we explore how to overcome the limitations of the architecture of Horváth et al. using a FPP that embeds a CNN-UM. The main advantage of the solution presented here is that each pixel in the FPP is capable of both sensing and processing, thus minimizing the delay introduced by the image transfer into the net. In addition, one single FPP can process layers with depths > 1. Finally, on the practical side, the size of the chip is quite convenient for embedded applications.

III. The Focal Plane Processor

Most smart CMOS image sensors, with different pixel types and different levels of intelligence, follow a conventional architecture where sensing —which is analog— is physically separated from processing, i. e. most of the intelligence is far from the sensor. Hence, all input data, most of which is irrelevant, must be encoded in digital format prior to processing. This fact stresses the system requirements regarding memory and computing resources. In addition to this, it creates significant bottlenecks in the dataflow.

In this work we have employed a commercial smart CMOS sensor called Eye-RIS, shown in Fig. 1(b) [10]-[11]. The sensor chip contains a FPP whose core is a CNN-UM. It contains 176 × 144 sensing and processing cells, locally interconnected with a 3 × 3-cell neighborhood. The cell's output is a piecewise nonlinear function with an operation range of $\pm x$. Table I shows a selection of operations available in the FPP. Most operations are fully-parallel realized in the analog domain. This has important advantages: for a moderate accuracy, analog signal processing blocks are less power-consuming usually faster and more compact than digital building blocks. Consequently, the FPP is capable of performing complex image processing in an ultra-efficient manner, in terms of both speed and power. Specifically, the FPP used here, exhibits a computational power of 250 GOPS with a power consumption of 4mW per GOPS.

Table I Available operations in the FPP

Greyscale/binary operations	Memory transfers
Storage in LAM/LDM	From local to external memory
Convolution, addition, subtraction and scaling by a constant	Between local memories
Operations on a ROI defined by binary mask	Between pixels

IV. Convolutional Neural Networks

ConvNets are a class of deep learning networks that consists of one input and one output layers, and also multiple hidden layers. The hidden layers of a ConvNet typically consist of a series of stacked convolutional, activation and pooling layers. As an example, in Fig. 1(a) displays a ConvNet with a LeNet-based architecture with a stack of two types of layers. Each convolutional layer is further broken down into different feature maps. Here, each neuron (or cell) unit within a feature map is connected to regions of feature maps of prior layers, except the input layer. The connections are supported by filter banks, i. e. sets of weights. Each filter is defined by a kernel whose dimensions represent the receptive field of the neuron.

Let us consider a simple ConvNet that accepts a single channel (monochrome) image as input, with a receptive field of 3 × 3-pixel size. The output of a neuron located at position (i,j) in the first convolutional layer l, and at depth f will be given by the equation:

$$C_f^1(i,j) = \sum_{u,v=-1}^{+1} K_f^1(u-i, v-j) I(i,j) = \\ = K_f^1 * I(i,j) \equiv \mathrm{CONV}_f^1[I(i,j)] \quad (1)$$

where $I(i,j)$ is the input signal at pixel (i,j) and K_f^1 is the kernel f of the first convolution layer. In short notation $\mathrm{CONV}_f^1[I(i,j)]$ is the convolution at depth f in the layer 1. The output of the convolutional layer, $O_f^1(i,j)$ is then calculated using a nonlinear activation function $F(\cdot)$:

$$O_f^1(i,j) = F\{\mathrm{CONV}_f^1[I(i,j)]\} \quad (2)$$

For deeper layers the output of a neuron located at (i,j) in the convolutional layer l, at depth g is given by:

$$O_g^l(i,j) = F\left[\sum_{f=1}^{n_f} K_g^l * O_f^{l-1}(i,j)\right] = \\ = F\left\{\sum_{f=1}^{n_f} \mathrm{CONV}_g^l\left[O_f^{l-1}(i,j)\right]\right\} \quad (3)$$

where O_f^{l-1} is the output of the feature map f of layer $l-1$ and n_f is the total number of filters (or depth) of the convolutional layer $l-1$.

The output of each convolutional layer is further processed with a pooling layer used to reduce the dimensionality of the feature maps. Pooling layers are usually of the type Max or Average. Finally, the output layer is a fully connected layer —whose inputs are all the pixels of the previous layer.

V. Implementation of ConvNet operators on the FPP

A. Computing precision of the analog FPP

The major part of the processing realized in Eye-RIS FPP is performed in the analog domain. The main drawback is the moderate to low precision of grey-level (analog) operations, which are in the range of 6-8 bits, depending on the specific block. For example, local analog memories responsible for greyscale image samples storage, initially have an 8b-equivalent resolution, although they are affected by a limited retention time [13]. Nevertheless, although most ConvNets are trained in high-precision (up to 32 bits) it is also true that resolutions of 8 bits or less are enough at inference. In fact researchers have shown that even ternary weights (0, ±1) may be enough to operate ConvNets with high classification accuracy [14].

B. Image scaling

The output of a cell in the FPP is a signed analog value with an equivalent resolution of 8 bits. This means that, generally speaking, signal range in grayscale (analog) processing must be controlled. For example, if two arbitrary images with values within the range $[-x, x]$ are to be subtracted, the resulting values can be within the interval $[-2x, 2x]$, which is out of range. The solution is pre-scaling both input images by multiplying by ½ before performing the subtraction, so the result remains inside the permitted signal range. Similarly, for the implementation of Eqs. (3) and (4) images should be pre-scaled by factor $1/n_f$, where n_f is the number of feature maps of a convolutional layer.

C. Convolutions

The FPP can perform convolutional operations in the analog domain in all cells $C(i,j)$ in parallel. However, the size of the template (or kernel, or receptive field) that can be used is limited by hardware construction to 3 × 3 pixels. Notice that for a given feature map g in the layer l, the kernel K_g^l is constant, therefore, if we recall the distributive property of convolution the output of a neuron of the feature map can be calculated also as:

$$O_g^l(i,j) = F\left[K_g^l * \sum_{f=1}^{n_f} O_f^{l-1}(i,j)\right] = \\ = F\left\{\mathrm{CONV}_g^l\left[\sum_{f=1}^{n_f} O_f^{l-1}(i,j)\right]\right\} \quad (4)$$

The advantage of the last expression compared to Eq. (3) is that the output of one neuron in a layer can be obtained using one single convolutional operation, that can speed inference time.

The output of a neuron $O_f^l(i,j)$ in a convolutional layer is calculated using a nonlinear activation function. The most commonly employed is the ReLU function, defined as:

$$O_g^l(i,j) = \mathrm{ReLU}\left[C_f^l(i,j)\right] = \\ = \begin{cases} C_f^l(i,j) & \text{if} \quad C_f^l(i,j) \geq 0 \\ 0 & \text{if} \quad C_f^l(i,j) < 0 \end{cases} \quad (5)$$

Notice that the outputs of greyscale operations realized by the FPP are signed with the zero level in the center of the signal range (Fig. 1(c)). The positive part of the nonlinear function is the same than the ReLU function and can be implemented easily with the FPP. As the resolution of the FPP is 8 bits, the range of values of the output is [0, 255] and the zero level is at 127. Then, by subtracting the value 127 to the image stored in a LAM, and adding again the value 127 to the result will set the cells with negative values to 0 as required by Eq. (5).

D. Pooling Layers

Pooling layers used in ConvNets are used for down sampling the dimensions of feature maps. Most of them use Max pooling layers, which can be implemented in a CNN through the iterative application of 8 convolutions and summations as described in [9]. Here we propose instead the use of Average pooling layers. Average pooling is more efficient than Max pooling because it can be implemented with only one convolution and a multiplication by a constant. The Average pooling layers available in this FPP are:

$$B = \frac{1}{4}\begin{bmatrix}1 & 1 & 0\\ 1 & 1 & 0\\ 0 & 0 & 0\end{bmatrix} \quad \text{and} \quad B = \frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix} \quad (6)$$

because the only kernel size available is 3 × 3 pixels. Another limitation of using the FPP is that pooling can only be used with stride 1, that does not decrease the size of the feature maps. An alternative is not using image padding. Doing so the image dimensions are reduced by 2 pixels in each dimension each time the pooling operation is applied.

E. Fully Connected Layer

Because the FPP is based on a locally connected network it is not efficient for implementing the Fully Connected Layer (or dense layer) used at the output of the ConvNet. In any case, the dense layer represents only a processing time overhead of about 10% [9]. Moreover, dense layers with up to 256 hidden nodes can be seamlessly implemented in the digital co-processor of the Eye-RIS.

VI. Preprocessing a CNN on the FPP

A. Simulation of a LeNet-type ConvNet

Table II compares two ConvNets trained on the MNIST dataset, with an input image size of 28 × 28 pixels, in terms of accuracy. First row corresponds to the original LeNet architecture, and the second one is a version of the one presented in reference [9]. The accuracy of the version model is just 1% less than the standard LeNet, but using only 8

convolutions, and Average pooling instead of Max filters with stride 1. This is equivalent to using the kernels of Eq. (8) and no padding. Also, the number of nodes at the dense layer at the output of the ConvNet is reduced from the original 512 to 256 to take into account the resources of the Eye-RIS.

Table II Accuracy of ConvNet architectures

MODEL	1ST CONV	1ST POOL	2ND CONV	2ND POOL	DENSE	ACCUR. %
LeNet	K=20 5×5	Max 2×2 stride 1	K=50 5×5	Max 2×2 stride 1	500	98
This work[a]	K=4 3×3	Avg 2×2 stride 1	K=4 3×3	Avg 2×2 stride 1	256	97

[a.] The model is a version of the one proposed in [9]

B. Algorithm for ConvNet inference and test on the FPP

Next we show one possible implementation of an algorithm for testing the LeNet architecture devised in Fig. 1(a), using the parameters of Table II and Eqs. (3) to (6).

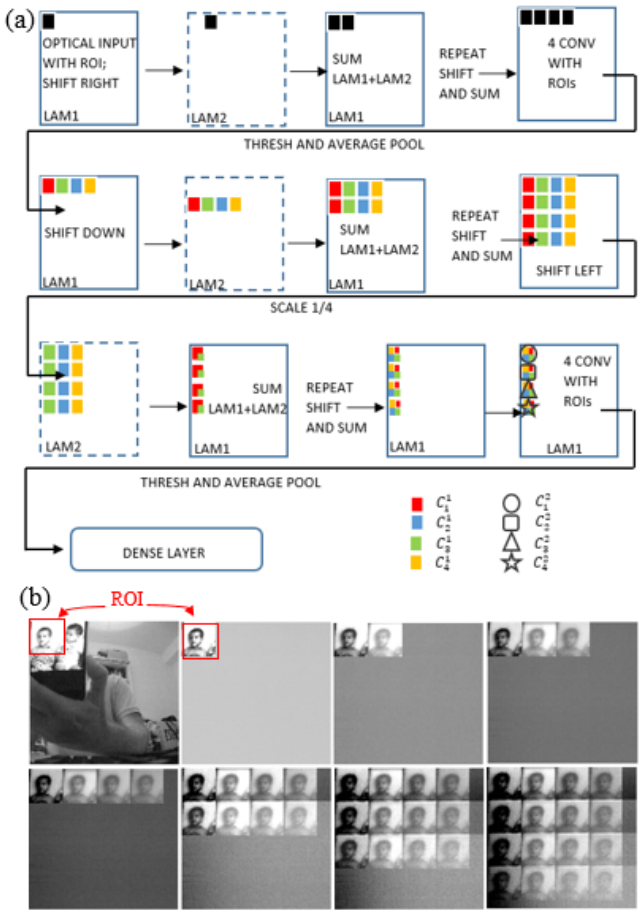

Fig. 2. (a) Graphical visualization of an algorithm for implementing the LeNet-type ConvNet described in Table II on the FPP. (b) Real test used for timing the speed of one forward pass .Note that althought the sensor array is 177 x 144 pixels in size the images have been cropped for better visualization. The time required for preprocessing all the layers except the dense layer is less than 300µs.

The algorithm is based on the following FPP capabilities:

- The FPP can sense images of 176 × 144 pixels while the input image size used on the LeNet architecture is of 28 × 28 pixels. Image acquisition can be done within a ROI of 28 × 28 pixels (see Table I).
- The content of a LAM in a specific cell can be shifted to any of the eight neighbours (LEFT, RIGHT, TOP, BOTTOM and the diagonals). Iterative shifting is possible.
- Two LAMs are available at each of the 176 × 144 pixels (LAM1 and LAM2 in Fig. 2). They are employed to greyscale image storage. The size of the sensing ROI is a fraction of the size of the complete array. Then the LAMs of those pixels that do not belong to the ROI are unused. Using iteratively image shifting operations, the unused pixels can be used for storing different images in the equivalent LAM location.

The proposed algorithm is graphically represented in Fig. 2(a). The first convolutional layer of the ConvNet is computed using Eq. (3) and the second convolutional layer using Eq. (4). Fig. 2(b) shows some partial results of testing the algorithm in the FPP. The performance was evaluated only in terms of speed, hence, the kernels were not optimized for the required 8-bits precision. Instead, random Gaussian kernels were used.

VII. CONCLUSIONS

We have reported the successful implementation of the majority of operations required for ConvNet inferencing on a single-chip CNN-based FPP that only uses linear B templates. Although the tests covered only an optimized LeNet ConvNet with just 8 convolutions, the FPP has the potential for processing networks with a larger number of convolutional layers and features maps, only limited by the sensor array/LAM size and the number of LAMs. An advantage of the FPP is that the speed for inference of a ConvNet like the one described here is independent of the array size. In other words, a larger CNN array can be used for processing larger images without changing the processing time. Timing tests show that the FPP can perform one image inference (one forward pass) in less 300µs. This is almost 2 orders of magnitude lower than the time required for real-time applications (@ 30fps, 33ms per frame). Future work will explore the implementation of the dense layer in the digital coprocessor and the optimization of the network using low-precision weights.

ACKNOWLEDGMENTS

The authors want to acknowledge funding from MCIU/AEI/ERDF-EU through projects PGC2018-101538-A-I00 and RTI2018-097088-B-C31. Financial support from the program ASECTI and Plan Propio-UCA Ref. 18INPPPR05 are also acknowledged. Also from EU H2020 MSCA through Project ACHIEVE-ITN (Grant No 765866) and the US Office of Naval Research through Grant No. N00014-19-1-2156 .

REFERENCES

[1] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "Cudnn: efficient primitives for deep learning," *arXiv preprint arXiv:1410.0759*, 2014.

[2] F. Ortega-Zamorano, J. M. Jerez, D. U. Munoz, R. M. Luque-Baena, and L. Franco, "Efficient implementation of the backpropagation

algorithm in FPGAs and microcontrollers," in 2015 *IEEE transactions on neural networks and learning systems*, vol. 27, no. 9, pp. 1840-1850.

[3] D. Moloney, "Embedded deep neural networks:'The cost of everything and the value of nothing'," in 2016 *IEEE Hot Chips 28 Symposium (HCS)*, Aug 2016, pp. 1-20.

[4] D. Monroe, "Neuromorphic computing gets ready for the (really) big time," in *Communications of the ACM*, vol. 57, no. 6, 2014, pp. 13-15.

[5] C. D. Schuman, T. E. Potok, R. M. Patton, J. D. Birdwell, M. E. Dean, G. S. Rose, and J. S. Plank, "A survey of neuromorphic computing and neural networks in hardware," *arXiv preprint arXiv:1705.06963*, 2017.

[6] L. O. Chua and T. Roska, "Cellular neural networks and visual computing: foundations and applications. Cambridge University Press, 2002.

[7] Á. Zarándy, A. Horváth, and P. Szolgay, "CNN Technology-Tools and Applications," in 2018 *IEEE Circuits and Systems Magazine*, vol. 18, no. 2 , Secondquarter 2018, pp. 77-89.

[8] G. Liñán Cembrano, A. B. Rodríguez Vázquez, R. Carmona Galán, F. J. Jiménez Garrido, S. C. Espejo Meana, and R. Domínguez Castro, "A 1000 FPS at 128× 128 vision processor with 8-bit digitized I/O,p" in 2004 IEEE Journal of Solid-State Circuits, vol. 39, no. 7, 2004, pp. 1044-1055.

[9] A. Horváth, M. Hillmer, Q. Lou, X. S. Hu, and M. Niemier, "Cellular neural network friendly convolutional neural network - CNNs with CNNs," in 2017 *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Mar 2017, pp. 145-150.

[10] A. Rodríguez-Vázquez, J. Fernández-Berni, J.A. Leñero-Bardallo, I. Vornicu, R. Carmona-Galán, "CMOS Vision Sensors: Embedding Computer Vision at Imaging Front-Ends". IEEE Circuits and Systems Magazine, Vol. 2ndQuarter 2018, pp. 90-107, April 2018.

[11] A. Rodríguez-Vázquez et al., "The Eye-RIS CMOS Vision System," in H. Casier. M. Steyaert M. and A. H. M. Van Roermund, Eds. Analog Circuit Design. Springer, Dordrecht, 2008.

[12] [Online] https://teledyne-anafocus.com/products/eye-ris-vision-system-on-chip/

[13] S. J. Carey and P. Dudek, "Vision chip with high accuracy analog S2I cells," *14th International Workshop on Cellular Nanoscale Networks and their Applications (CNNA)*, pp. 1-2, Notre Dame, IN, 2014.

[14] F Li, B. Zhang, and B. Liu. "Ternary weight networks," in *arXiv preprint arXiv:1605.04711*, 2016.