CNP: An FPGA-based Processor for Convolutional Networks

NEW YORK UNIVERSITY

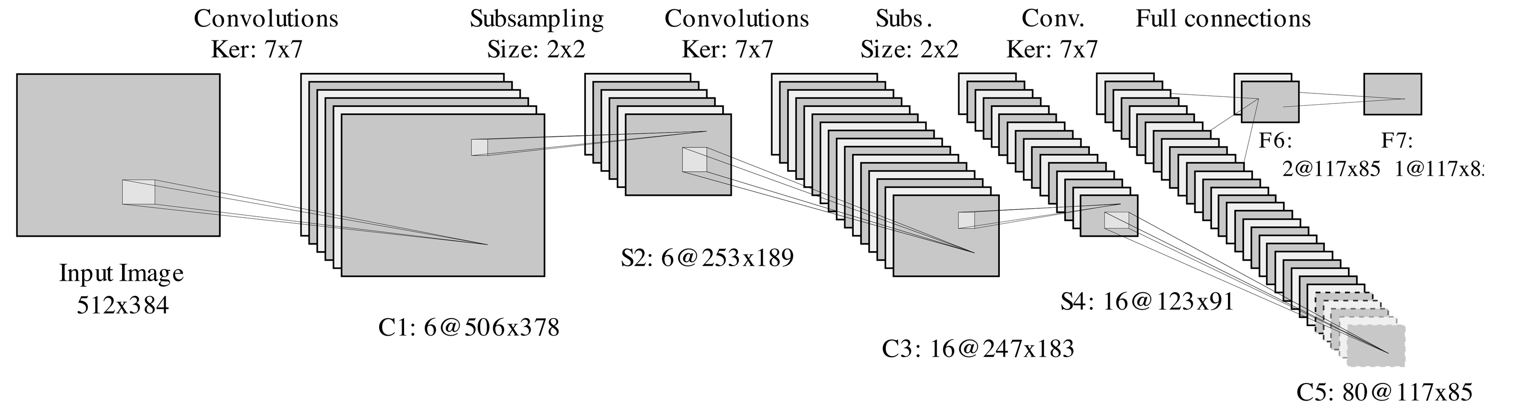
Clément Farabet[1], Cyril Poulet[1], Jefferson Y. Han[2] and Yann LeCun[1]

[1] *Courant Institute, New York University*
[2] *Perceptive Pixel Inc.*

Overview

We present an efficient implementation of convolutional networks (ConvNets) on a DSP-oriented Field Programmable Gate Array (FPGA). The implementation exploits the parallel structure of ConvNets and takes full advantage of multiple hardware multiply-accumulate units on the FPGA. Our system uses a single FPGA with an external memory module. A compiler software was implemented, to convert trained ConvNets into code for the ConvNet Processor (CNP). This design can be used for low-power, lightweight embedded vision systems for micro-UAVs and other small robots.

CNP to compute ConvNets



Convolutional Networks (ConvNets) are:

- made of a feed-forward, bio-inspired architecture consisting of multiple linear convolution filters interspersed with point-wise non-linear squashing functions and pooling functions,
- trainable to perform detection, recognition and segmentation on raw images.

To be run on the CNP, a ConvNet must be:

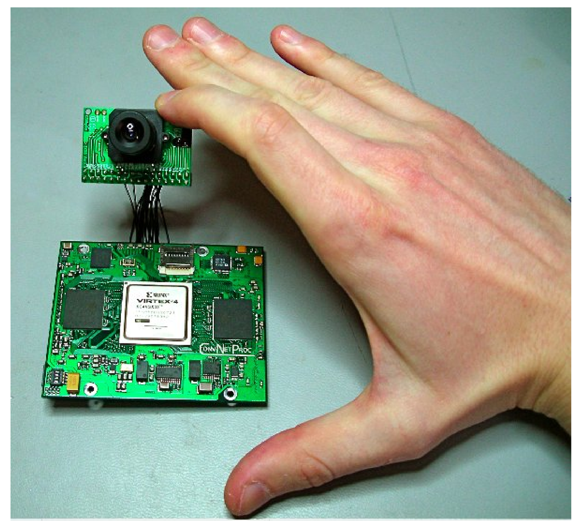
- defined and trained on a conventional machine, using a learning library (e.g. `Lush`, `EBLearn`)
- extracted and compiled into the proper sequence of calls to run on the CNP (we designed a compiler for this task)

Custom Design

Our current system:

- fits in a single FPGA + an external memory!
- has been integrated onto a 7x8cm printed circuit board!
- only draws up to 15W during peak computations!

In a UAV, the CNP could be used as the main vision sensor (the camera sensor is handled by the CNP) to detect and track obstacles.



References

[1] R. G. Shoup, "Parameterized convolution filtering in a field programmable gate array," in *Selected papers from the Oxford 1993 international workshop on field programmable logic and applications on More FPGAs*. Oxford, United Kingdom: Abingdon EE&CS Books, 1994, pp. 274–280.

[2] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, November 1998.

[3] Y. LeCun, "Generalization and network design strategies," in *Connectionism in Perspective*, R. Pfeifer, Z. Schreter, F. Fogelman, and L. Steels, Eds. Zurich, Switzerland: Elsevier, 1989, an extended version was published as a technical report of the University of Toronto.

[4] C. Garcia and M. Delakis, "Convolutional face finder: A neural architecture for fast and robust face detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 11, pp. 1408–1423, 2004.

[5] Y. LeCun and L. Bottou, "Lush reference manual," Tech. Rep., 2002, code available at http://lush.sourceforge.net. [Online]. Available: http://lush.sourceforge.net

[6] M. Osadchy, Y. LeCun, and M. Miller, "Synergistic face detection and pose estimation with energy-based models," *Journal of Machine Learning Research*, vol. 8, pp. 1197–1215, May 2007.

[7] Y. LeCun, U. Muller, J. Ben, E. Cosatto, and B. Flepp, "Off-road obstacle avoidance through end-to-end learning," in *Advances in Neural Information Processing Systems (NIPS 2005)*. MIT Press, 2005.

[8] R. Hadsell, A. Erkan, P. Sermanet, J. Ben, K. Kavukcuoglu, U. Muller, and Y. LeCun, "A multi-range vision strategy for autonomous offroad navigation," in *Proc. Robotics and Applications (RA'07)*, 2007.

[9] J. Cloutier, E. Cosatto, S. Pigeon, F. Boyer, and P. Y. Simard, "Vip: An fpga-based processor for image processing and neural networks," in *Fifth International Conference on Microelectronics for Neural Networks and Fuzzy Systems (MicroNeuro'96)*, Lausanne, Switzerland, 1996, pp. 330–336.

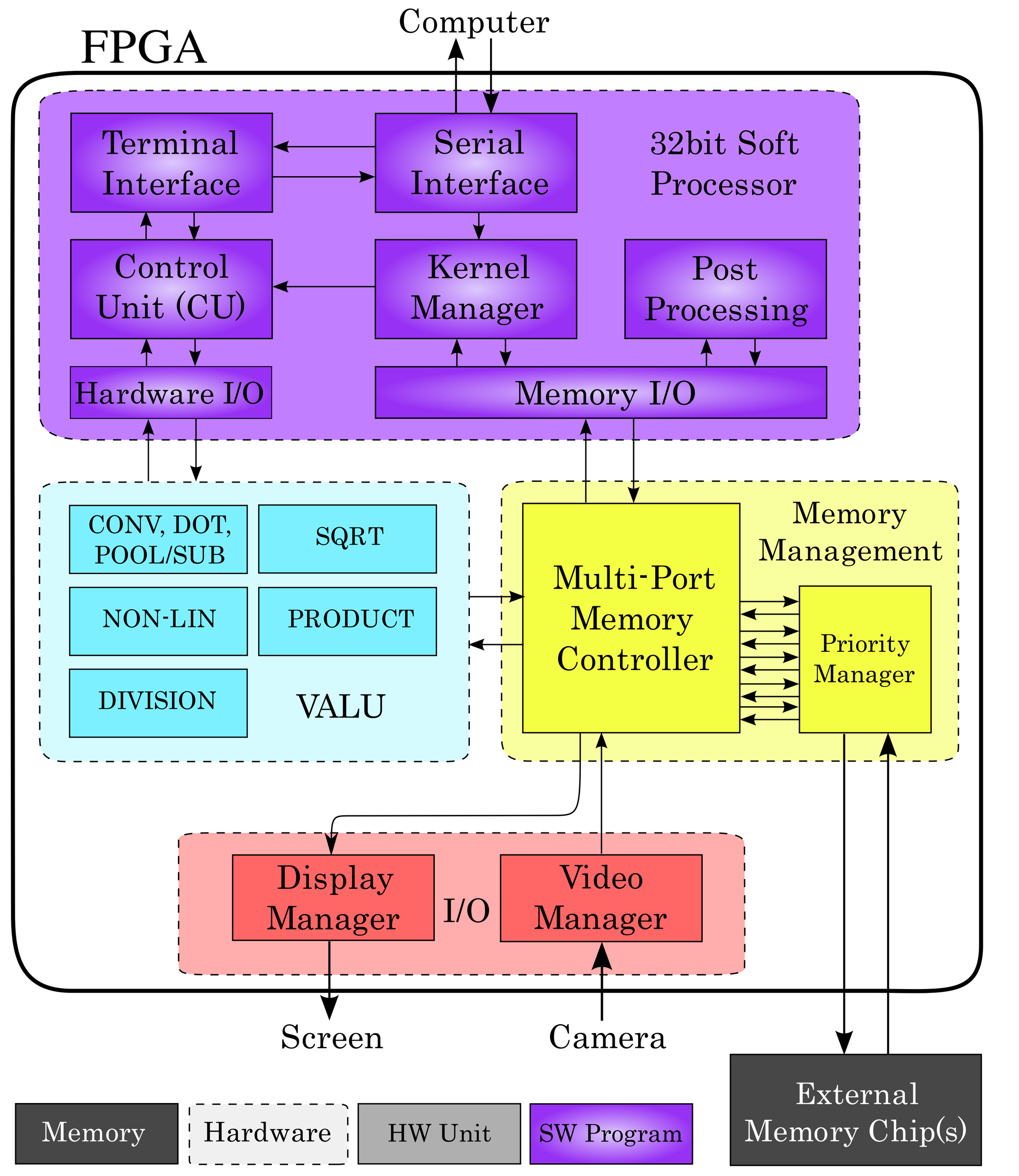
CNP: Architecture

The FPGA

① Our system fits in a single FPGA with built-in hardware multiply-accumulate units,
① Its performance is defined by the bandwidth to the external memory chip.

Vectorial ALU

Implements ConvNet-specific operations:
① All these operations are vectorial (work on streams), and compute in the same time for a given input size,
① These instructions are geared towards ConvNets, or more generally vision systems in which the bulk of computations is spent on convolutions,
① The instructions are:
- 2D convolution,
- dot product between n 2D planes and a vector (n dims),
- point-wise non-linear mapping,
- spatial (2D) pooling / subsampling,
- square root of a vector/matrix,
- product between vectors/matrices,
- division of a vector/matrix by another



Soft Processor

Adds a layer of abstraction to the system: a program on the CPU acts as a micro-program for the VALU, and implementing a particular ConvNet simply consists in reprogramming this processor. A few programs running on this CPU:
① Control Unit & Hardware I/O: control the structure of the ConvNet to be computed by sequencing operations of the VALU. This is a software-emulated control unit !
① Post processing operations for object detection include non-max suppression, calculation of centroids of activities, and other functions that are not worth implementing in hardware.

Memory Controller

Provides an abstraction over the memory:
① Instructions in the ALU operate on streams of data,
① Streams are handled by a complex memory controller, that allows different units of the system to read/write from/ to the external memory asynchronously,
① A dedicated hardware arbiter (priority manager) is used to multiplex/demultiplex access to the high bandwidth external memory chip.

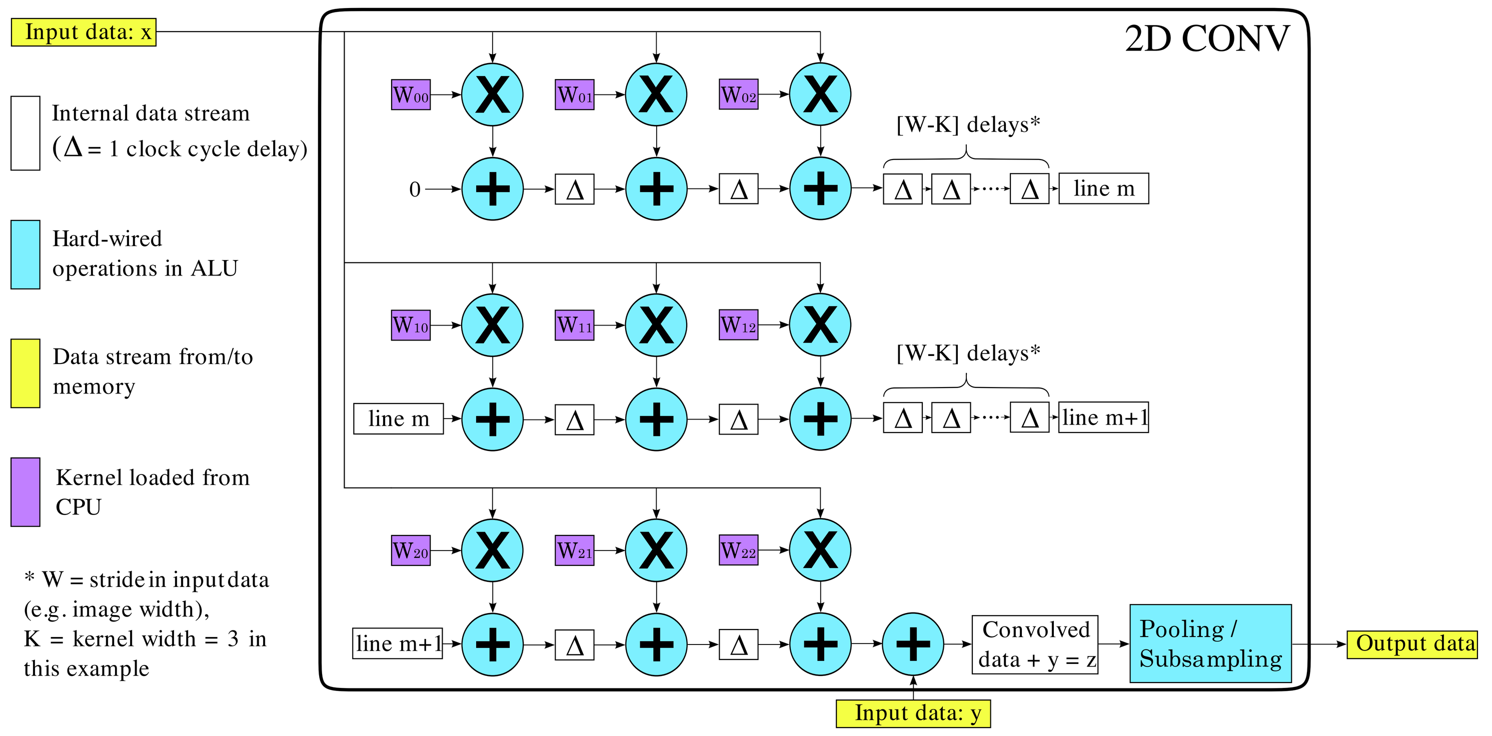
135 GOP/s peak with 13x13 kernels **10W** on average **>1000** convs / sec on 640x480 inputs

VALU: Main Instructions

2D Convolver / Pooling
This instruction performs a 2D convolution [1] on streaming data, and applies spatial pooling at the output. At each clock cycle, z_{ij} is computed according to the formula:

$$z_{ij} = y_{ij} + \sum_{m=0}^{K-1} \sum_{n=0}^{K-1} x_{i+m,j+n} ker_{mn},$$

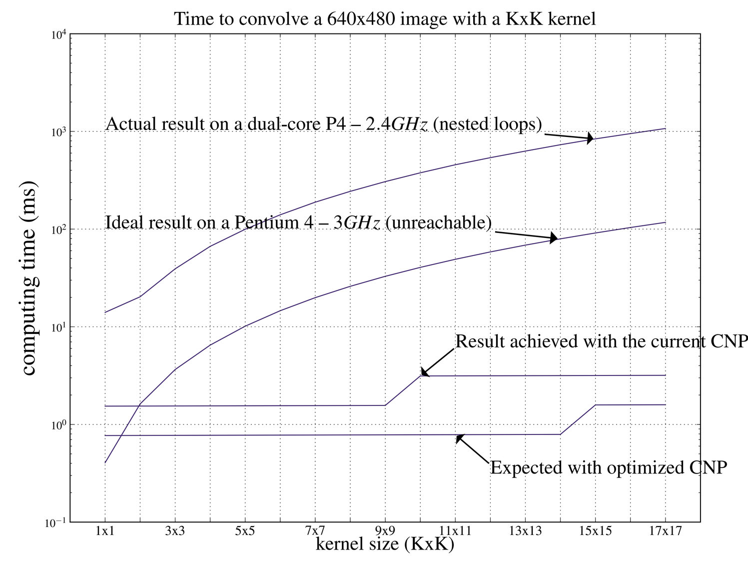
x: input plane,
ker: $K \times K$ convolution kernel,
y: plane accumulated to output, z: output plane.

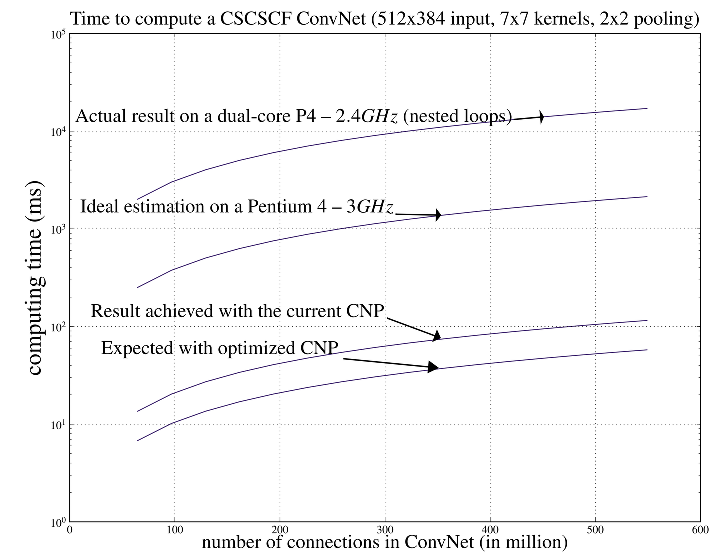


Non-linear mapping
The point-wise non-linearity is implemented as a piecewise approximation of the hyperbolic tangent function: $g(x) \approx A.\tanh(B.x)$, with the following constraint (to use shifts and adds instead of multipliers):

$$g(x) = a_i x + b_i \quad for \quad x \in [l_i, l_{i+1}] \quad and \quad a_i = \frac{1}{2^m} + \frac{1}{2^n} \quad m, n \in [0, 5].$$

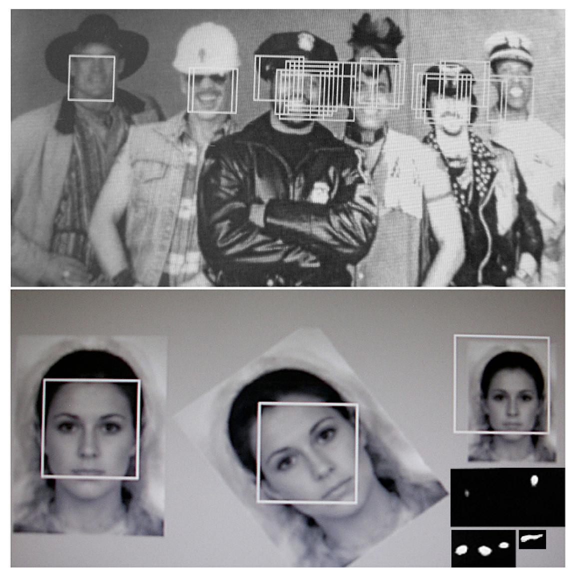
Performance





An Application: Face Detection

A ConvNet was trained on a dataset of faces and non-faces according to the method described in [2]. Its architecture—number of layers, feature maps—is given in the first figure.



The dataset: 45,000 images—30,000 used for training, 15,000 for testing—50% faces, and 50% random images (non faces).

The CNP computes this ConvNet—image acquisition, pre-processing, layers of the network, post-processing and classification—at 10fps for a 512x384 input image size and 3 scales.

The two pictures show the output of the system when running the face detection ConvNet. These images are generated on the fly by an asynchronous display manager.