# Hardware Accelerated Convolutional Neural Networks for Synthetic Vision Systems

Clément Farabet[1,2], Berin Martini[2], Polina Akselrod[2], Selçuk Talay[2], Yann LeCun[1] and Eugenio Culurciello[2]
[1] The Courant Institute of Mathematical Sciences and Center for Neural Science, New York University, USA
[2] Electrical Engineering Department, Yale University, New Haven, USA

*Abstract*— **In this paper we present a scalable hardware architecture to implement large-scale convolutional neural networks and state-of-the-art multi-layered artificial vision systems. This system is fully digital and is a modular vision engine with the goal of performing real-time detection, recognition and segmentation of mega-pixel images. We present a performance comparison between a software, FPGA and ASIC implementation that shows a speed up in custom hardware implementations.**

## I. Introduction

Micro-robots, UAVs, imaging sensor networks, wireless phones, and other embedded vision systems all require low cost and high-speed implementations of synthetic vision system capable of recognizing and categorizing objects in a scene. Virtually all recent synthetic vision algorithms targeting general recognition problems use one or more layers of filter banks organized hierarchically to report some degree of invariance in position, angle and size of the image features [1]. Examples are the SIFT algorithm, [2], bio-inspired algorithms modeling the mammalian visual system [3], and deep architectures [4], [5] using multi-layer neural networks. Synthetic vision algorithms with multiple layers of features extraction and learned parameters perform the best [1]. Learning provides significant performance improvements when specific targets are known a priori [4], [6]. Convolutional neural networks (ConvNets) are a synthetic vision architecture that embeds all these features. ConvNets are feed-forward neural networks with multiple layers of convolution filters and non-linearities [4], [6].

In this paper we present a scalable hardware architecture for large-scale multi-layered synthetic vision systems based on large parallel filter banks. This hardware can also be used to accelerate the execution (and partial learning) of recent vision algorithms like SIFT and HMAX [2], [3]. This system is a data-flow vision engine that can perform real-time detection, recognition and localization in mega-pixel images processed as pipelined streams. The system was designed with the goal of providing categorization of an arbitrary number of objects, while consuming ten times less than a bench-top or laptop computer (target: $< 10W$).

## II. Convolutional Neural Networks

Convolutional neural networks or ConvNets are a special kind of neural networks that take advantage of the locality of data in images to reduce the number of parameters needed to process large images. An example of a ConvNet is given in Figure 1. In this figure we report a network used for generic object recognition of N classes. Such a network has been successfully used to classify different objects in numerous applications [5].

ConvNets have several advantages as a front-end for synthetic vision systems that perform object categorization tasks. First, they operate with local receptive fields by performing convolutions: they share weights in the convolution matrices, so large images can be processed with a reduced set of weights. This is important as the number of weights in the network is thus not proportional to the input image (i.e. the final processing network size is fixed for a specific task.). Second, spatial subsampling/pooling is used to hierarchically reduce the input data size at each step of nonlinear computation. Replicating a small, local receptive field extracts elementary features from a large input, while sub-sampling the result reduces the effect of distortion and scale. Combining these features produces higher-order features that have very good shift, scale and distortion invariance, a typical feature of high-level mammalian vision systems [3]

An important aspect of ConvNets is that all of their parameters can be learned from the data to be modeled. In the network presented in Figure 1 for instance, all the weights from the filter banks, pooling functions, and also from the classifier are learned at the same time, by using stochastic gradient descent on labeled dataset. When compared to hand-tweaked feature extractors, ConvNets are more compact, and amenable to general purpose recognition tasks.

## III. System Implementation

A fully-digital coded hardware implementation of a scalable ConvNet [4] has been developed and implemented. Small analog versions of ConvNets have been implemented, but at the time were not able to scale [7]. We believe a fully-digital implementation with current FPGA and ASIC technologies is the easiest way to get a software-compatible object-recognition networks, that is easy to setup and operate, use reduced power consumption and provides high numeric precision. The entire system is coded in hardware description languages (HDL), and is targeted for ASIC synthesis or programmable hardware like FPGAs. The design is a custom single instruction multiple data (SIMD) processor based on a 32 bit CPU with hardware-accelerated instructions tailored to ConvNet operations. These operations are highly optimized and make use of the parallelism available in hardware.

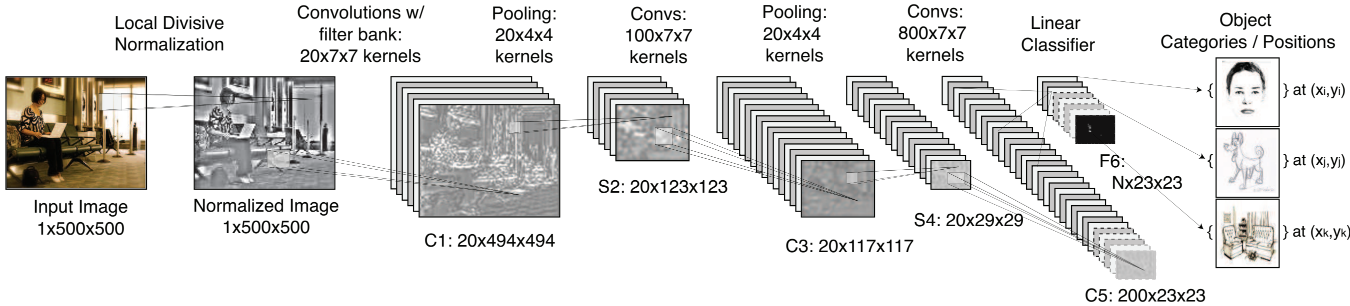


Fig. 1. Architecture of a typical convolutional network for object recognition. This implements a convolutional feature extractor and a linear classifier for generic N-class object recognition. All filter kernels sizes are mentioned above the networks, and are all learned from labeled data. The bottom numbers report the size of image maps at each layer of the network. The network can be computed on arbitrary large input images, producing a classification for each 40x40 sub-window (size of the training data).

Our first implementation of a ConvNet processor on FPGA was developed on a Xilinx Virtex-4 SX35 FPGA board. The FPGA was connected to external QDR-SRAM memory in a custom designed printed circuit board [8]. The custom board operates at 200MHz with a 72bit wide bus to the memory and 7.2GB/s of memory bandwidth, and was able to implement convolution arrays of up to 13x13 with the 192 multipliers of the Virtex-4 FPGA SX35. This system achieved very good performance, but had limitations in its architecture that lead to suboptimal usage of the available bandwidth.

### A. Architecture

The *second-generation architecture* proposed in this paper was designed to increase data throughput by adding multiple parallel vector processing units and allowing individual streams of data to operate seamlessly within processing blocks. A schematic summary of the *Stream Processor* system is presented in Figure 2. The main components of our system are: (1) a *Control Unit* CPU, (2) multiple parallel *ALUs/Streaming Operators*, and (3) a *Memory Interface Streaming Engine*.

A *general purpose CPU* acts as a flexible Control Unit for our system by controlling the configuration bus. Other modules in the system are connected to a global configuration bus, which allows for run-time reconfiguration of any parameters in the system: from the connections between processing tiles, to the 2D data dimensions mapped to the external memory.

The *ALUs* are independent processing tiles laid out on a two-dimensional grid. Each tile is composed of: a *Global Router*, *Local Routers*, a *Streaming Operator*. The ALUs in Figure 2 have been simplified for clarity: each tile actually contains four local routers to stream data from/to any of their four neighbors, and one global router to stream data from/to global data lines. The local and global routers are configured at run-time to allow arbitrary routes of data streams in and out each tile.

The operators in the ALU are fully pipelined to produce one result per clock cycle. The ALUs implement all the typical macroscopic operators required to compute layers of bio-inspired models, and more precisely ConvNets: 1) 2D convolver (implemented in the FPGA by the dedicated multipliers), 2) dot products between a vector and multiple 2D planes, 3) spatial pooling (image subsampling), 4) arbitrary non-linear mappings (such as: sigmoid, hyperbolic tangent, square root, rectification), 5) element-wise division of a vector by another, 6) element-wise multiplication of a vector by another. The 2D convolutions in a ConvNet use 80 to 90% of the total amount of computations performed by the network, so large arrays of multiply-accumulate units are needed to accelerate the system [8].

We chose Q8.8 as coding of numbers in the network, after estimating the influence of quantization noise. The pipelined implementation of the convolution operation in hardware (as described in [8]), computes equation 1 at every clock cycle.

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

In equation 1 $x_{ij}$ is a value in the input plane, $w_{mn}$ is a value in a $K \times K$ convolution kernel, $y_{ij}$ is a value in a plane to be combined with the result, and $z_{ij}$ is the output plane. Both the kernel and the image are streams loaded from the memory, and the filter kernels can be pre-loaded in local caches concurrently to another operation: each new pixel thus triggers $K \times K$ parallel operations. The convolution pipeline is 32bit wide, to keep full precision between successive accumulations.

All the non-linearities in neural networks are implemented with a piecewise linear approximation operator, as described by 2. The piecewise mapping is performed by a hardware mapper, which streams the input data through a cascade of simple linear mappers. Each mapper is configurable at run-time, so many different functions can be programmed and used in the network. By using coefficient $a_i$ in equation 3, all the linear mappings can be implemented with shifts and adds only.

$$g(x) = a_i x + b_i \quad for \quad x \in [l_i, l_{i+1}] \quad (2)$$

$$a_i = \frac{1}{2^m} + \frac{1}{2^n} \quad m, n \in [0, 5]. \quad (3)$$

The data lines to and from memory are handled by a *Multiport DMA Streaming Engine* specifically designed for image manipulations. The streaming engine interfaces any kind of memory module (internal or external), and offers Nx16bit asynchronous read/write ports on the other side,

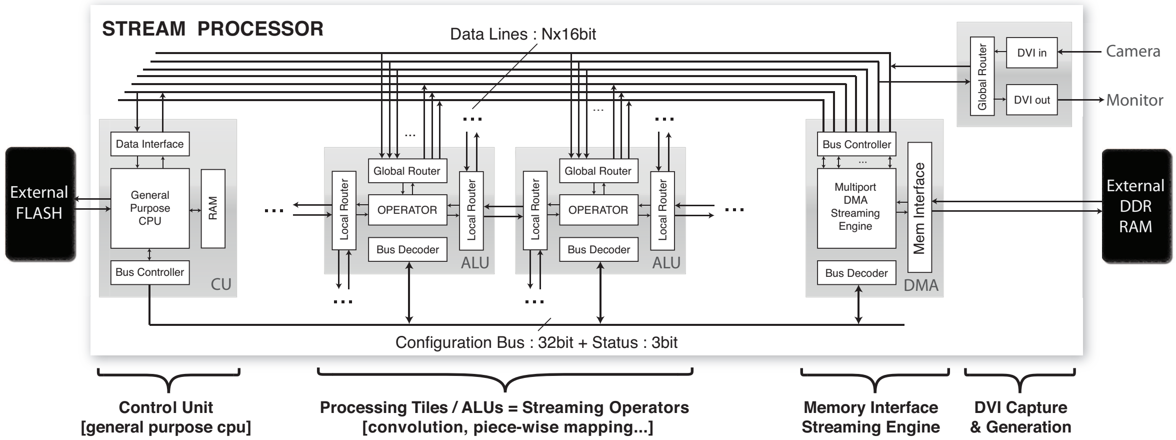


Fig. 2. Overview of the hardware ConvNet system. A CPU was augmented with multiple full-custom ALUs tailored to ConvNet operations, and a fast streaming memory interface. ALUs are organized on a 2D grid: they can stream data to their closest neighbors, and to the global lines connected to the memory interface.

which allows multiple simultaneous streams from/to the same memory locations, even if at different data rates. A dedicated arbiter is used as hardware *Memory Interface* to multiplex and demultiplex access to the external memory with high bandwidth. Subsequent buffers on each port insure continuity of service on a port while the others are utilized.

This module is the foundation of our system: it allows any other module in the entire design to read/write any image as a continuous stream of pixels from/to memory without any concern for the others. This allows to perform multiple operations (such as convolutions) on multiple image maps at the same time.

In our current system, the ALUs are synchronous to the memory bus clock, which gives the following relationship between the memory bandwidth $B$, the number of possible parallel data transfers $N$ and the bits per pixel $P$: $N = B/P$. For example $P = 16bit$ and $B = 128bit/cyc$ allows $N = 7$ simultaneous transfers.

## B. Operation

The typical execution of an instruction on this system is the following: 1) the CPU configures each tile to be used for the computation and each connection between the tiles and their neighbors and/or the global lines, by sending a configuration command to each of them, 2) it configures the streaming engine to prefetch the data to be processed, and to be ready to write the results, 3) when the streaming engine is ready, it triggers the streaming out, 4) each ALU processes its respective incoming streaming data, and passes the results to another tile, or back to the streaming engine, 5) the CPU is notified of the end of operations when the streaming engine has completed. The behavior of each port in the Streaming Engine can be configured separately by using the configuration bus. The configuration consists of the 2D offsets and 2D dimensions of an image in memory (which is viewed as two-dimensional by the streaming engine).

Prior to being run on the Stream Processor, a ConvNet has to be trained offline, on a regular computer, and then converted to a compact representation that can be interpreted by an embedded program (running on the CPU) to generate the control/configuration for the system. Special software was developed to unify these two steps. This software, written in pure `C++`, is used to train then execute ConvNets on a variety of embedded platforms, with or without hardware acceleration. The library, called Nrgizer, provides a high-level modular and scalable implementation of ConvNets (similar to Lush [1]).

A typical setup would be as such: 1) Nrgizer is used on a computer to train a ConvNet for a particular task, using some dataset, 2) the trained network is saved and quantized to the Stream Processor's Q8.8 coding, 3) it is sent to a running version of Nrgizer on the Stream Processor, and saved locally (flash memory), 4) it is then interpreted on the CPU, by using all the ALUs/Streaming capabilities that are implemented in Nrgizer as low-level routines (software Control Unit), 5) embedded Nrgizer has full access to the data, and can perform post-processing operations for object detection, such as non-max suppression, calculations of centroids of activities (attention), and other functions that do not necessitate an hardware implementation.

# IV. Results

Figures 3 and 4 report a performance comparison between a laptop CPU, a FPGA implementation, and a future ASIC implementation for the computation of the ConvNet presented in Figure 1. This network is composed of a non-linear normalization layer, 3 convolutional layers, 2 pooling layers, and a linear classifier. The convolutional layers and pooling layers are followed by non-linear activation units (hyperbolic tangent). Overall, it possesses 920 KxK learned kernels, 40 4x4 learned subsampling kernels, and N 200 dimension classification vectors. For a 500x500 input image and $K = 7$, the network has 435 Million linear connections (multiply and accumulate operations).

Figure 3 shows the frames per second versus input image size with a fixed 9x9 convolution filter ($K = 9$) for the whole

[1]Lisp Universel SHell: `http://lush.sourceforge.net`

ConvNet, and 5 output classes ($N = 5$). When the input image size varies, the network adapts the sizes of all its internal maps accordingly, producing an output map with a size linearly related to the input size.

Figure 4 reports frames per second vs convolution filter sizes, assuming the ConvNet uses the same filter size in all three layers, an input image of 500x500 pixels, and other parameters as mentioned above. When the kernel sizes vary, the internal maps sizes vary accordingly.

The CPU data was measured from our compiled library Nrgizer (optimized C code, GNU C compiler) on a Core 2 Duo 2.4GHz Apple Macbook PRO laptop operating at 90W, the FPGA data was measured on a Xilinx Virtex-4 SX35 operating at 200MHz and 15W with 3 data lines used in parallel (to and from the external memory) [8], and the ASIC data is simulation data gathered from Tezzaron 3D process. A conservative estimate of the projected ASIC speed is 400MHz (speeds of up to $> 1$ GHz are possible) with 8 ALU convolvers running in parallel, and 10 data lines. The projected power consumption of the ASIC is 1W.

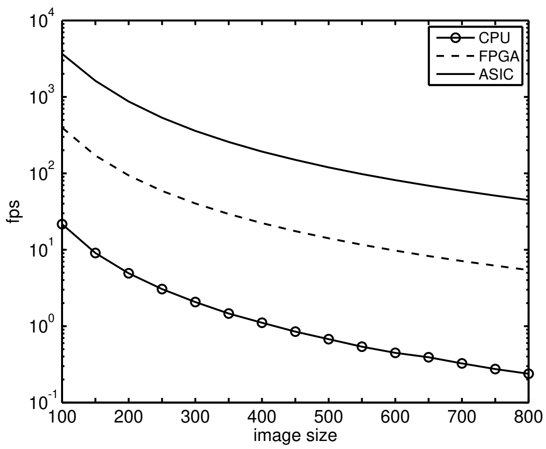


Fig. 3. Frames/s from a ConvNet similar to Fig.1 vs the size of input images and using convolutional filters of 9x9.

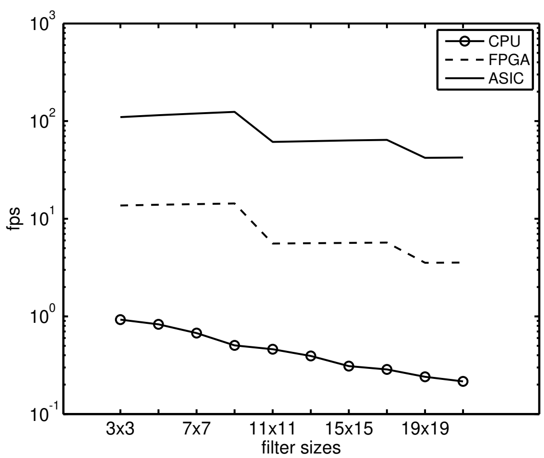


Fig. 4. Frames/s from a ConvNet similar to Fig.1 vs the size of convolution filters for a fixed image size of 500x500.

As we can see from these results an ASIC system or a modern Virtex 6 FPGA can run the ConvNet system in real time ($> 30fps$) with filters up to 21x21 in size, and images approaching 1 mega-pixels.

Nrgizer has been tested on standard datasets: *Small NORB*, *MNIST* and the *UMASS* face dataset. For *NORB*, 85% recognition rate was achieved on the unknown dataset, for *MNIST*, 95% and for *UMASS*, 98%. The same test were conducted on the Stream Processor with fixed-point representation (Q8.8), and the results were, respectively: 85%, 95% and 98%.

## V. Conclusion

We report the design of a hardware accelerated ConvNet system that is capable of running in real time with low power consumptions, while providing performance that is better than conventional laptop computers. This architecture is demonstrated in multiple FPGA implementations. Future work will include implementation in a high-performance ASIC system capable of delivering real-time operation on 1 mega-pixel images with 1W of power.

While our current FPGA implementation can perform medium complexity tasks such as face detection/tracking in real-time, the ASIC implementation will open the doors to more complex and generic recognition tasks. Multiple object detection [5] or online learning for adaptive robot guidance [9] are tasks that will be largely improved by this system.

## Acknowledgment


This work was partially supported by NSF grant ECCS-0901742, by ONR MURI BAA 09-019, DARPA NeoVision2 program BA 09-58.


## References


[1] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, "What is the best multi-stage architecture for object recognition?" in *Proc. International Conference on Computer Vision (ICCV'09)*. IEEE, 2009.

[2] S. Lazebnik, C. Schmid, and J. Ponce, "Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories," in *CVPR '06: Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 2169–2178.

[3] T. Serre, L. Wolf, S. Bileschi, M. Riesenhuber, and T. Poggio, "Robust object recognition with cortex-like mechanisms," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 3, pp. 411–426, 2007.

[4] 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.

[5] Y. LeCun, F.-J. Huang, and L. Bottou, "Learning methods for generic object recognition with invariance to pose and lighting," in *Proceedings of CVPR'04*. IEEE Press, 2004.

[6] M. Ranzato, F. Huang, Y. Boureau, and Y. LeCun, "Unsupervised learning of invariant feature hierarchies with applications to object recognition," in *Proc. Computer Vision and Pattern Recognition Conference (CVPR'07)*. IEEE Press, 2007.

[7] E. Säckinger, B. Boser, J. Bromley, Y. LeCun, and L. D. Jackel, "Application of the ANNA neural network chip to high-speed character recognition," *IEEE Transaction on Neural Networks*, vol. 3, no. 2, pp. 498–505, March 1992.

[8] C. Farabet, C. poulet, J. Han, and Y. LeCun, "CNP: An FPGA-based Processor for Convolutional Networks," in *International Conference on Field Programmable Logic and Applications*. Prague: IEEE, September 2009.

[9] R. Hadsell, P. Sermanet, M. Scoffier, A. Erkan, K. Kavackuoglu, U. Muller, and Y. LeCun, "Learning long-range vision for autonomous off-road driving," *Journal of Field Robotics*, vol. 26, no. 2, pp. 120–144, February 2009.