frontiers in
NEUROSCIENCE

ORIGINAL RESEARCH ARTICLE
published: 12 April 2012
doi: 10.3389/fnins.2012.00032

Comparison between frame-constrained fix-pixel-value and frame-free spiking-dynamic-pixel convNets for visual processing

Clément Farabet[1,2]*, Rafael Paz[3], Jose Pérez-Carrasco[4], Carlos Zamarreño-Ramos[4], Alejandro Linares-Barranco[3]*, Yann LeCun[1], Eugenio Culurciello[2]*, Teresa Serrano-Gotarredona[4] and Bernabe Linares-Barranco[4]

[1] *Computer Science Department, Courant Institute of Mathematical Sciences, New York University, New York, NY, USA*
[2] *Laboratoire d'Informatique Gaspard-Monge, Université Paris-Est, Équipe A3SI, ESIEE Paris, Champs sur Marne, Marne-la-Vallée, France*
[3] *Robotic and Technology of Computers Group, University of Seville, Seville, Spain*
[4] *Instituto de Microelectrónica de Sevilla, IMSE-CNM-CSIC, Sevilla, Spain*

Edited by:
Tobi Delbruck, Institute for Neuroinformatics, Switzerland

Reviewed by:
Tobi Delbruck, Institute for Neuroinformatics, Switzerland
Anton Civit, University of Seville, Spain
Zhengming Fu, Advanced Micro Devices, USA

****Correspondence:***
Clément Farabet, Computer Science Department, Courant Institute of Mathematical Sciences, New York University, 715 Broadway, 12th Floor, New York, NY 10003, USA.
e-mail: cfarabet@nyu.edu; www.clement.farabet.net;
Alejandro Linares-Barranco, Robotic and Technology of Computers Lab, University of Seville, ETSI Informática, Av. Reina Mercedes s/n, 41012-Sevilla, Spain.
e-mail: alinares@atc.us.es;
Eugenio Culurciello, Weldon School of Biomedical Engineering, Purdue University, 206 S. Martin Jischke Drive, Room 2031, West Lafayette, IN 47907, USA.
e-mail: euge@purdue.edu

Most scene segmentation and categorization architectures for the extraction of features in images and patches make exhaustive use of 2D convolution operations for template matching, template search, and denoising. Convolutional Neural Networks (ConvNets) are one example of such architectures that can implement general-purpose bio-inspired vision systems. In standard digital computers 2D convolutions are usually expensive in terms of resource consumption and impose severe limitations for efficient real-time applications. Nevertheless, neuro-cortex inspired solutions, like dedicated Frame-Based or Frame-Free Spiking ConvNet Convolution Processors, are advancing real-time visual processing. These two approaches share the neural inspiration, but each of them solves the problem in different ways. Frame-Based ConvNets process frame by frame video information in a very robust and fast way that requires to use and share the available hardware resources (such as: multipliers, adders). Hardware resources are fixed- and time-multiplexed by fetching data in and out. Thus memory bandwidth and size is important for good performance. On the other hand, spike-based convolution processors are a frame-free alternative that is able to perform convolution of a spike-based source of visual information with very low latency, which makes ideal for very high-speed applications. However, hardware resources need to be available all the time and cannot be time-multiplexed. Thus, hardware should be modular, reconfigurable, and expansible. Hardware implementations in both VLSI custom integrated circuits (digital and analog) and FPGA have been already used to demonstrate the performance of these systems. In this paper we present a comparison study of these two neuro-inspired solutions. A brief description of both systems is presented and also discussions about their differences, pros and cons.

Keywords: convolutional neural network, address-event-representation, spike-based convolutions, image convolutions, frame-free vision, FPGA, VHDL

1. INTRODUCTION

Conventional vision systems process sequences of frames captured by video sources, like webcams, camcorders (CCD sensors), etc. For performing complex object recognition algorithms, sequences of computational operations are performed for each frame. The computational power and speed required makes it difficult to develop a real-time autonomous system. But brains perform powerful and fast vision processing using small and slow cells (neurons) working in parallel in a totally different way. Vision sensing and object recognition in the mammalian brain is not performed frame by frame. Sensing and processing are performed in a continuous way, spike by spike, without any notion of frames.

The visual cortex is composed by a set of layers (Shepherd, 1990; Serre, 2006), starting from the retina. The processing starts beginning at the time the information is captured by the retina. Although cortex has feedback connections, it is known that a very fast and purely feed-forward recognition path exists in the visual cortex (Thorpe et al., 1996; Serre, 2006).

In recent years significant progress has been made toward the understanding of the computational principles exploited by the visual cortex. Many artificial systems that implement bio-inspired software models use biological-like (convolution based) processing that outperform more conventionally engineered machines (Neubauer, 1998). These systems run at low speeds when implemented as software programs on conventional computers. For real-time solutions direct hardware implementations of these models are required. However, hardware engineers face a large hurdle when trying to mimic the bio-inspired layered structure and the massive connectivity within and between layers. A growing number of research groups world-wide are mapping some of these computational principles onto both real-time spiking hardware through the development and exploitation of the so-called AER (Address-Event-Representation) technology, and real-time streaming Frame-Based ConvNets on FPGAs.

ConvNets have been successfully used in many recognition and classification tasks including document recognition (LeCun et al., 1998a), object recognition (Huang and LeCun, 2006; Ranzato et al., 2007; Jarrett et al., 2009), face detection (Osadchy et al., 2005), and robot navigation (Hadsell et al., 2007, 2009). A ConvNet consists of multiple layers of filter banks followed by non-linearities and spatial pooling. Each layer takes as input the output of previous layer and by combining multiple features and pooling over space, extracts composite features over a larger input area. Once the parameters of a ConvNet are trained, the recognition operation is performed by a simple feed-forward pass.

The simplicity of the feed-forward pass has pushed several groups to implement it as custom hardware architectures. Most of ConvNet hardware implementations reported over the years are for the frame-constrained fix-pixel-value version, as they map directly from the software versions. The first one was the ANNA chip, a mixed high-end, analog-digital processor that could compute 64 simultaneous 8×8 convolutions at a peak rate of 4.109 MACs (multiply-accumulate operations per second; Boser et al., 1991; Säckinger et al., 1992). Subsequently, Cloutier et al. proposed an FPGA implementation of ConvNets (Cloutier et al., 1996), but fitting it into the limited-capacity FPGAs available at those times required the use of extremely low-accuracy arithmetic. Modern DSP-oriented FPGAs include large numbers of hard-wired multiply-accumulate units that can greatly speed up compute-intensive operations, such as convolutions. The frame-constrained system presented in this paper takes full advantage of the highly parallel nature of ConvNet operations, and the high-degree of parallelism provided by modern DSP-oriented FPGAs. Achieved peak rates are in the order of 10^{11} MACs.

On the other hand, Frame-free Spiking-Dynamic-Pixel ConvNets compute in the spike domain. No frames are used for sensing and processing the visual information. In this case, special sensors are required with a spike-based output. Spike-based sensors and processors typically use AER (Address-Event-Representation) in order to transmit the internal state and/or results of the neurons inside a chip or FPGA.

AER was originally proposed almost twenty years back in Mead's Caltech research lab (Sivilotti, 1991). Since then AER has been used fundamentally in vision (retina) sensors, such as simple light intensity to frequency transformations (Culurciello et al., 2003; Posch et al., 2010), time-to-first-spike coding (Ruedi et al., 2003; Chen and Bermak, 2007), foveated sensors (Azadmehr et al., 2005), spatial contrast (Costas-Santos et al., 2007; Massari et al., 2008; Ruedi et al., 2009; Leñero-Bardallo et al., 2010), temporal contrast (Lichtsteiner et al., 1998; Posch et al., 2010; Leñero-Bardallo et al., 2011), motion sensing and computation, (Boahen, 1999), and combined spatial and temporal contrast sensing (Zaghloul and Boahen, 2004). But AER has also been used for auditory systems (Chan et al., 2007), competition and winner-takes-all networks (Chicca et al., 2007; Oster et al., 2008), and even for systems distributed over wireless networks (Teixeira et al., 2006). After sensing, we need Spiking Signal Event Representation techniques capable of efficiently processing the signal flow coming out from the sensors. For simple per-event heuristic processing and filtering, direct software based solutions can be used (Delbrück, 2005, 2008). Other schemes rely on look-up table re-routing and event repetitions followed by single-event integration (Vogelstein et al., 2007). Alternatively, we can find some pioneering work in the literature aiming at performing convolutional filteringon the AER flow produced by spiking retinas, (Vernier et al., 1997; Choi et al., 2005), where the shape of the filter kernel was hard-wired (either elliptic or Gabor). Since 2006, working AER Convolution chips have been reported with arbitrary shape programmable kernel of size up to 32×32 pixels pre-loaded onto an internal kernel-RAM (Serrano-Gotarredona et al., 2006, 2008; Camuñas-Mesa et al., 2011, 2012). This opens the possibility of implementing in AER spiking hardware generic ConvNets, where large number of convolutional modules with arbitrary size and shape kernels are required.

In this paper we present, discuss and compare two different neuro-cortex inspired approaches for real-time visual processing based on convolutions: Frame-based fix-pixel-value and Frame-free dynamic-pixel-spiking ConvNet Processing hardware.

Section 2 describes generic ConvNets and their structure. Section 3 briefly describes frame-free ConvNet types of implementations, and Section 4 describes a frame-constrained FPGA implementation. Implemention details will be given in a very concise manner, so the reader can grasp the main ideas behind each implementation. For more detailed descriptions the reader is refer to the corresponding references. Finally, Section 5 provides a comparison of both cases indicating pros and cons of each.

2. STRUCTURE OF GENERIC ConvNets

Figure 1 shows a typical hierarchical structure of a feed-forward ConvNet. Convolutional Networks (LeCun et al., 1990, 1998a), or ConvNets, are trainable multi-stage architectures composed of multiple stages. The input and output of each stage are sets of arrays called *feature maps*. For example, if the input is a color image, each feature map would be a 2D array containing a color channel of the input image (for an audio input each feature map would be a 1D array, and for a video or volumetric image, it would be a 3D array). At the output, each feature map represents a particular feature extracted at all locations on the input tolerating degrees of deformations and sizes.

Each stage is composed of three layers: a *filter bank layer*, a *non-linearity layer*, and a *feature pooling layer*. A typical ConvNet is composed of one, two, or three such 3-layer stages, followed by a classification module. Each layer type is now described for the case of image recognition.

2.1. FILTER BANK LAYER - *F*

The input is a 3D array with n_1 2D *feature maps* of size $n_2 \times n_3$, and coordinates(x_i, y_i), with $i = 1,\ldots n_1$. Let's call each input feature map $f_i = (x_i, y_i)$, with $x_i = 1,\ldots n_2$ and $y_i = 1,\ldots n_3$. The output is also a 3D array composed of m_1 feature maps of size $m_2 \times m_3$ and coordinates (X_j, Y_j) with $j = 1,\ldots m_1$. Let's call each output feature map $F_j = (X_j, Y_j)$, with $X_j = 1,\ldots m_2$ and $Y_j = 1,\ldots m_3$. A trainable filter (kernel) w_{ij} in the filter bank has size $l_1 \times l_2$ and connects input feature map f_i to output feature map F_j. The

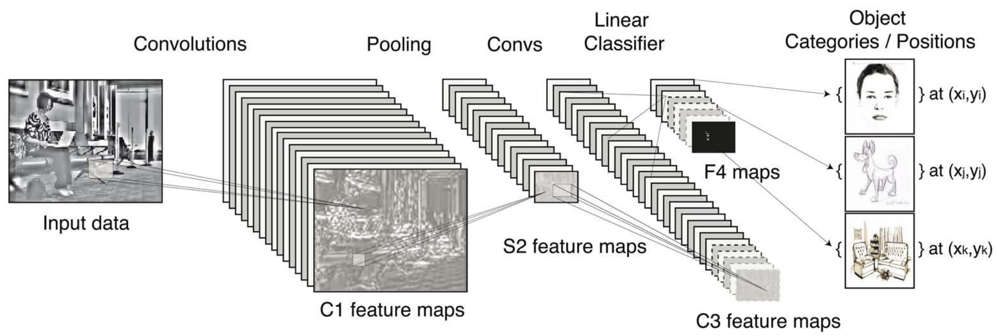


FIGURE 1 | Architecture of a typical convolutional network for object recognition. This implements a convolutional feature extractor and a linear classifier for generic N-class object recognition. Once trained, the network can be computed on arbitrary large input images, producing a classification map as output.

module computes $F_j = b_j + \Sigma_i w_{ij} * f_i$ where $*$ is the 2D convolution operator and b_j is a trainable bias parameter. Each filter detects a particular feature at every location on the input. Hence spatially translating the input of a feature detection layer will translate the output but leave it otherwise unchanged.

2.2. NON-LINEARITY LAYER

In traditional ConvNets this simply consists of a point wise tanh() sigmoid function applied to each site(X_j, Y_j). However, recent implementations have used more sophisticated non-linearities (Lyu and Simoncelli, 2008; Pinto et al., 2008).

2.3. FEATURE POOLING LAYER

This layer treats each feature map separately. In its simplest instance, called P_A, it computes the average values over a neighborhood in each feature map. This results in a reduced-resolution output feature map which is robust to small variations in the location of features in the previous layer. The average operation is sometimes replaced by a max P_M. Traditional ConvNets use a point wise *tanh*() after the pooling layer, but more recent models do not.

Supervised training is performed using a form of stochastic gradient descent to minimize the discrepancy between the desired output and the actual output of the network. All the filter coefficients in all the layers are updated simultaneously by the learning procedure. The gradients are computed with the back-propagation method. Details of the procedure are given in LeCun et al. (1998a), and methods for efficient training are detailed in LeCun et al. (1998b).

3. FRAME-FREE SPIKING-DYNAMIC-PIXEL ConvNets

In frame-free spiking ConvNets the retina sensor pixels generate spikes autonomously. Pixel activity changes continuously, as opposed to frame-based systems, where the pixel value is frozen during each frame time. Such spikes are sent to projection fields in the next layer, and the contribution of each spike is weighted by a 2D spatial filter/kernel value w_{ij} over the projection field. In the next layer pixels, incoming weighted spikes are accumulated (integrated) until a pixel fires its own spike for the next layer, and so on. Each pixel in any Convolution Module represents its state by its instantaneous spiking activity. Consequently, each pixel at any layer has to be present at any time and its state cannot be fetched in and out as in Frame-based approaches. This is the main drawback of this approach: all ConvModules have to be there in hardware and hardware resources cannot be time-multiplexed.

Adapting ConvNets to Spiking Signal Event-based representations yields some very interesting properties. The first one is the very reduced latency between the input and output event flows of a spiking convolution processor. We call this the "*pseudo-simultaneity*" between input and output visual flows. This is illustrated by the example at the end of Section 3.

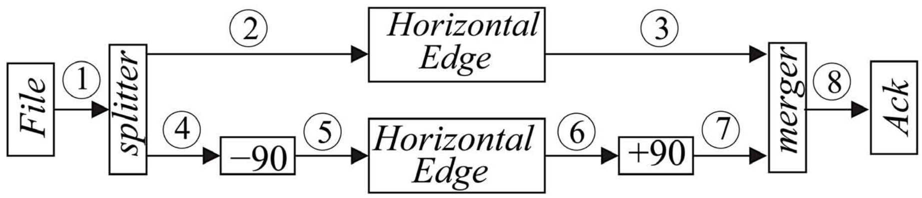
The second interesting property of implementing Spiking Event Convolutions (or other operators, in general) is its modular scalability. Since event flows are asynchronous, each AER link between two convolutional modules is independent and needs no global system level synchronization.

And the third interesting property of spike-based hardware, in general, is that since processing is per-event, power consumption is, in principle, also per-event. Since events usually carry relevant information, power is consumed as relevant information is sensed, transmitted, and processed.

Next we describe briefly three ways of computing with spiking ConvNets. First, we briefly describe an event-based simulation software tool for emulating such spiking AER hardware systems. Second, we briefly summarize some programmable kernel VLSI implementations. And third, similar FPGA implementations are discussed.

3.1. SOFTWARE SIMULATOR

A behavioral event-driven AER simulator has been developed for describing and studying generic AER systems (Pérez-Carrasco, 2011). Such simulator is very useful for designing and analyzing the operation of new hardware systems combining existing and non-existing AER modules. Modules are user-defined and they are interconnected as defined by a netlist, and inputs are given by stimulus files. The simulator was written in C++. The netlist uses only two types of elements: AER modules (instances) and AER links (channels). AER links constitute the nodes of the netlist in an AER system. Channels represent point-to-point connections. Splitter and merger instances are used for spreading or merging links. **Figure 2** shows an example system and its text file netlist description with 7 instances and 8 channels. Channel 1 is a source channel. All its events are available *a priori* as an




```
%First, we declare sources to the system
%                SOURCES         SOURCES DATA
sources            {1}               {s1}

%Next, we declare blocks
%NAME              IN-CHANNELs    OUT-CHANNELSPARAMES       STATES
splitter           {1}            {2,4}    {params1}        {state1}
h_sobel            {2}            {3}      {params2}        {state2}
imrotate90         {4}            {5}      {params3}        {state3}
h_sobel            {5}            {6}      {params4}        {state4}
imrotate-90        {6}            {7}      {params5}        {state5}
merger             {3,7}          {8}      {parmas6}        {state6}
ack_only           {8}            {8}      {params7}        {state7}
```

FIGURE 2 | Example netlist and its ASCII file netlist description.

input file. These events can be pre-recorded by a real AER retina (Lichtsteiner et al., 1998; Posch et al., 2010; Leñero-Bardallo et al., 2011). Each instance is defined by a line. Instance operation is described by a user-defined function. Channels are described by lists of events. Once the simulator has finished, there will be a list of time-stamped events for each node. Each event is defined by 6 values (T_{pR}, T_{Rqst}, T_{Ack}, a, b, c). The first 3 are timing parameters and the other three are open user-defined parameters that the instances interpret and interchange. Usually, a and b are the event address (x, y) and c its sign. T_{Rqst} is the time when an event *Rqst* was generated and T_{Ack} when it was acknowledged. T_{pR} is the time of creation of an event (before communicating or arbitrating it out of its source module). The simulator scans all channels looking for the earliest unprocessed T_{pR}. This event is processed: its T_{Rqst} and T_{Ack} are computed and the state of the event destination modules are updated. If this creates new events, they are added to the end of the corresponding links event lists, and the list is re-sorted for indreasing T_{pR}. Then the simulator looks again for the earliest unprocessed T_{pR}, and so on.

3.2. VLSI IMPLEMENTATION

Reported VLSI implementations of AER spiking ConvModules (either mixed-signal, Serrano-Gotarredona et al., 2006, 2008; or fully digital, Camuñas-Mesa et al., 2011, 2012) follow the floor plan architecture in **Figure 3**, where the following blocks are shown: (1) array of lossy integrate-and-fire pixels, (2) static RAM that holds the stored kernel in 2's complement representation, (3) synchronous controller, which performs the sequencing of all operations for each input event and the global forgetting mechanism, (4) high-speed clock generator, used by the synchronous controller, (5) configuration registers that store configuration parameters loaded at startup, (6) left/right column shifter, to properly align the

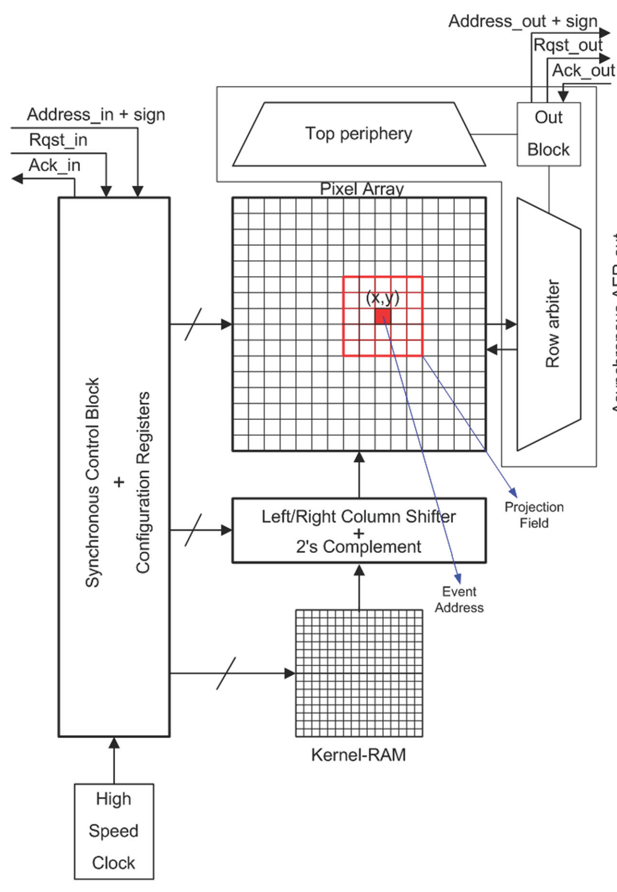


FIGURE 3 | Architecture of the convolution chip.

stored kernel with the incoming event coordinates, (7) AER-out, asynchronous circuitry for arbitrating and sending out the output address events generated by the pixels, and (8) for the digital version a 2's complement block is required to invert kernel data before adding them to the pixels, if an input event is negative. When an input event of address (x, y) is received, the controller copies row after row the kernel values from the kernel-RAM to the corresponding pixel array rows (the projection field), as indicated in **Figure 3**. Then all pixels within this projection field update their state: they add/subtract the corresponding kernel weight depending on event and weight signs. When a pixel reaches its positive or negative threshold, it signals a signed output event to the peripheral arbiters, which send its address and sign out. Parallel to this per-event processing, there is a global forgetting mechanism common for all pixels: pixel values are decremented (if they are positive) or incremented (if they are negative) triggered by a global periodic signal. This implements a constant leak of fixed rate that discharges the neurons, allowing the ConvModule to capture dynamic reality with a time constant in the order of this leak. A more formal mathematical justification of this event-driven convolution operation can be found elsewhere (Serrano-Gotarredona et al., 1999).

3.3. FPGA IMPLEMENTATION

Figure 4 shows the block diagram of an FPGA spike-based convolver. A serial peripheral interface (SPI) is used to communicate with a USB microcontroller in order to allow to change the configuration from a laptop (Kernel matrix, kernel size, forgetting period, and forgetting quantity). The circuit in the FPGA can be divided into the following parallel blocks:

- A 64 × 64 array of memory cells: the matrix is implemented using a block of dual-port RAM in the FPGA. Each position of the RAM is 8-bit length.
- Kernel memory: The kernel is stored also in the internal RAM of the FPGA in an 11 × 11 matrix with 8-bit resolution.
- Conv state machine: Each input event corresponds to the address of a pixel. Centered on this address, the kernel is added to the memory matrix, which is used to save the state of the convolution cells. If any of the modified cells reaches a value higher than a global programmable threshold (Th), an output event with this cell address is queued to be sent through the AER output bus, and the cell is reset.
- Forgetting mechanism. A configurable forgetting circuitry is also present in the architecture. The forgetting is based on a programmable counter that accesses the memory matrix periodically in order to decrease its values by a constant.
- Memory arbiter. The 64 × 64 cell memory matrix is a shared resource between the forgetting circuitry and the convolution state machine. Therefore, a memory arbiter is required.
- FIFO and AER output state machine: A 16 event first-input-first-output buffer is used to store the outgoing events before they are transmitted by the state machine using the asynchronous protocol.
- SPI State Machine. This controller is in charge of receiving kernel size and values, forgetting period and amount to forget. The system is configured and controlled through a computer running MATLAB.

The system has been implemented in hardware in a Virtex-6 FPGA. A VHDL description of this ConvModule with 64 × 64 pixels and kernels of size up to 11 × 11 has been used to program different ConvModule arrays into a Virtex-6 FPGA, together with the corresponding inter-module communication and event routing machinery. The internal structure of commercial FPGAs with their internal memory arrangement and distribution is not optimum for implementing event-driven parallel modules. Nonetheless, it was possible to include an array of 64 Gabor filters, each with a specific scale and orientation to perform a V1 visual cortex pre-processing on event data coming out of a temporal difference retina (Zamarreño-Ramos, 2011; Zamarreño-Ramos et al., under review). **Table 1** summarizes the resources used by the Virtex-6.

3.4. EXAMPLE SYSTEM AND OPERATION

The example in **Figure 5** illustrates event-driven sensing and processing, and *pseudo-simultaneity*, on a very simple

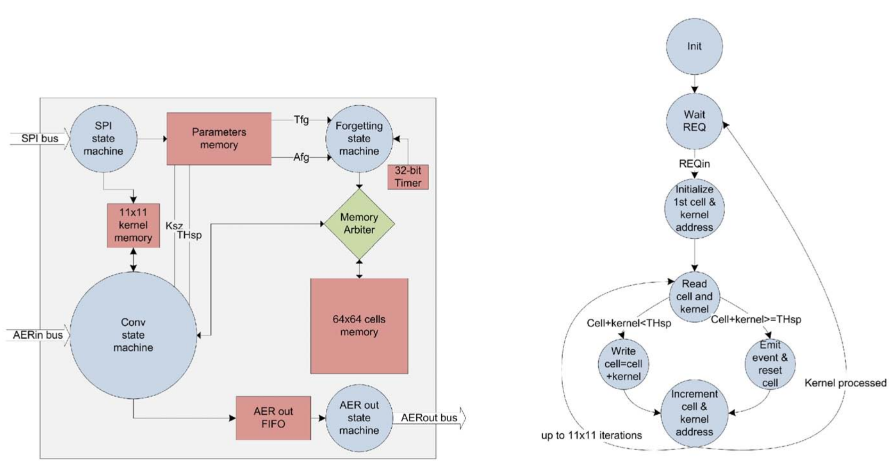


FIGURE 4 | Block diagram of the FPGA AER-based convolution processor (left) and its State Machine (right).

Table 1 | Frame-free FPGA resource consumption.

Resources of a Virtex6 LX240T	#Used
128 × 8-bit single-port block RAM	64
16 × 1-bit single-port read-only distributed RAM	64
16 × 16-bit dual-port distributed RAM	64
4096 × 8-bit single-port block RAM	64
4 × 4-bit single-port read-only distributed RAM	1
64 × 64-bit single-port read-only distributed RAM	1
2-33-bit adders/subtractors	2752
2-14-bit counters	1487
Flip-flops	91397
Finite-state-machines	1557
2-33-bit comparators	3274
1-32-bit multiplexors	25801
Slices registers	74987 out of 301440 (24%)
Slices LUTs	83521 out of 150720 (55%)
Occupied Slices	32720 out of 37680 (86%)
Block RAM36E1/FIFO	64 out of 416 (15%)
Block RAM18E1/FIFO	68 out of 832 (8%)

two-convolution setup. **Figure 5A** shows the basic setup. A 52 card deck is browsed in front of a motion sensitive AER retina (Leñero-Bardallo et al., 2011). **Figure 5B** shows a picture taken with a commercial camera with 1/60 sec (16.67 ms) exposure time. **Figure 5C** shows the events captured during a 5-ms time window, while a card with "clover" symbols is browsed. **Figure 5D** shows the instantaneous event rate for the whole event sequence when browsing the complete 52 card deck. Most cards are browsed in a 410-ms time interval, with peak event rate of about 8 Meps (mega events per second) computed on 10 μs time bins. The events produced by the retina are sent (event after event) to a first Event-Driven Convolution chip programmed with the kernel in **Figure 5E** to filter out noise and enhance shapes of a minimum size. The output events produced by this first Convolution chip are sent to a second Convolution chip programmed with the kernel in **Figure 5F**. This kernel performs crude template matching to detect "clover" symbols of a specific size and orientation. In order to perform more sophisticated size and pose invariant object recognition a full multi-stage ConvNet would be necessary. However, this simple example is sufficient to illustrate the pseudo-simultaneity property. The two-convolution system was simulated using the simulator described in Section 1 and using recorded event data taken from a real Motion Sensitive retina (Leñero-Bardallo et al., 2011) using an event data logger board (Serrano-Gotarredona et al., 2009). This event data logger board can record up to 500 k events with peak rates of up to 9 Meps. **Figure 5G** shows the retina events (red dots), the first convolution output events (green circles) and the second convolution output events (blue stars) in *y* vs. *time* projection, for a 85-ms time interval. One can see very clearly the events corresponding to 4 cards (numbered "1" to "4" in the figure). Cards "2" to "4" contain "clover" symbols that match the size and orientation of

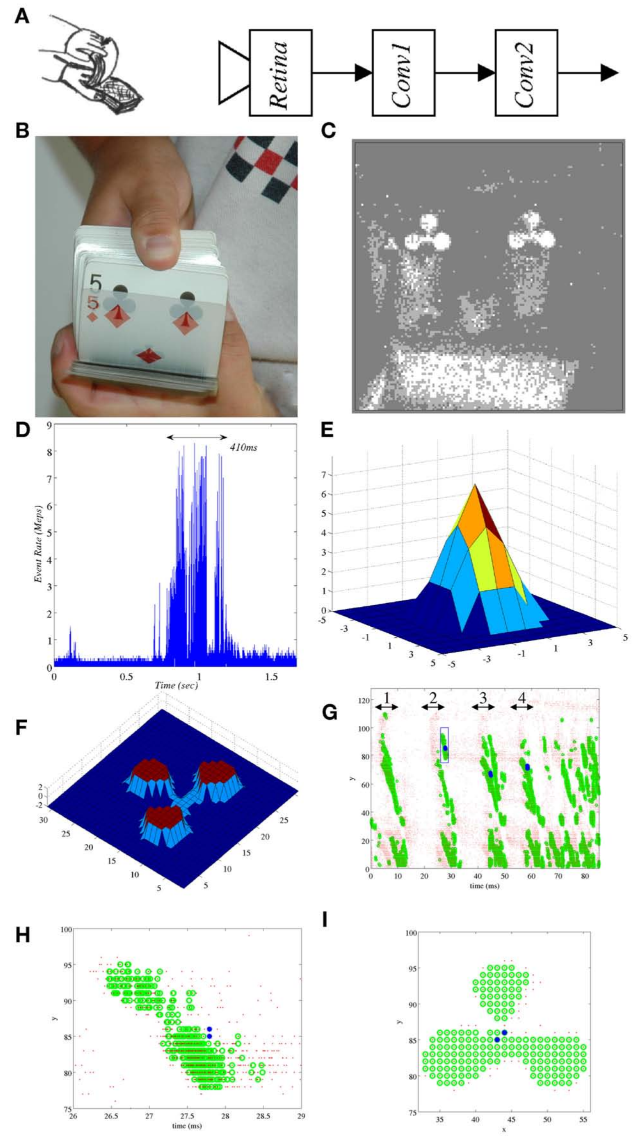


FIGURE 5 | Illustration of pseudo-simultaneity in fast event-driven recognition. (A) Feed-forward Two-Convolution system. **(B)** Photograph with commercial camera at 1/60 s. **(C)** Five milliseconds event capture from AER motion retina. **(D)** Event rate computed using 10 μs bins. **(E)** First pre-filtering Kernel. **(F)** Second template-matching kernel. **(G)** Events from real retina (red dots), simulated output of first filter (green circles), and simulated output of second filter (blue stars). **(H)** y/time zoom out. **(I)** x/y zoom out.

the kernel. **Figure 5G** includes a zoom box between 26 and 29 ms. The events inside this zoom box are shown in **Figure 5H** in *y* vs. *time* projection, and in **Figure 5I** in *y* vs. *x* projection. As one can see, between time 26 and 29 ms a clear "clover" symbol is present at the retina output (small red dots). The retina "clover" events range between 26.5 and 29 ms (2.5 ms duration). The output events of the first filter (green circles) range between time 26.5 and 28.5 ms (2.0 ms duration), which is inside the time window of the retina events. Consequently, retina and first convolution streams are simultaneous. The output events of the second Convolution (thick blue dots) are produced at time 27.8 ms (1.3 ms after the 1st retina "clover" event and 1.2 ms before the retina last "clover" event), which is during the time the retina is still sending out events of the "clover" symbol, and also while the first Convolution is still providing output events for this symbol. Note that the second convolution needs to collect a very large number of events before making a decision, because its kernel is very large. However, in a standard ConvNet with many ConvModules, kernels are usually much smaller and would require much less input events to start providing outputs, therefore also speeding up the whole recognition process, in principle. As can be seen in **Figures 5G,H**, clover symbol recognition is achieved even before the sensor has delivered all the events that form the symbol. All this illustrates quite nicely the *pseudo-simultaneity* property of frame-free event-driven systems.

This contrasts with the Frame-Constraint philosophy. Even if one has a very high-speed video camera, say 1 kframe/s, the system has first to acquire an image (which would take 1 ms), send it to a frame-constraint processing system (like the one described in Section 4), and assuming it can provide an output after another 1 ms, the recognition result would be available 2 ms after the start of sensing. Although these times are comparable to what is shown in **Figure 5H**, the sensing output and the processing output are sequential, they are not simultaneous. This is one key conceptual difference between the two approaches. To understand how this extrapolates to multiple layers, let us refer to **Figure 6**. At the top (**Figure 6A**) there is a 6-layer ConvNet feature extraction system for object recognition. Let us assume each layer contains a large number of feature extraction ConvModules, whose outputs are sent to each subsequent layer. Let us assume that we have a very fast Frame-based processing system per layer (as the one described in the next Section) and that it is capable of computing all feature maps within a layer in 1 ms. Let us assume also that we have a very fast sensor capable of providing a frame rate of 1 image/ms (1000 fps), and that the output of each stage can be transmitted to the next stage much faster than in 1 ms. Let us also assume that there is a sudden visual stimulus that lasts for about 1 ms or less. **Figure 6B** shows the timing diagram for the outputs x_i at each subsequent layer of a Frame-based implementation. The sudden stimulus happens between time 0 and 1 ms, and the sensor output is provided at time 1 ms. The first layer feature maps output is available at time 2 ms, the second at time 3 ms, and so on until the last output is available at time 6 ms. **Figure 6C** shows how the timing of the events would be in an equivalent six layer event-driven implementation. As in **Figure 5**, the sensor provides the output events simultaneously to reality, thus during the interval from 0 to 1 ms. Similarly, the 1st event-driven feature maps x_1 would be available during the same interval, and so on for all subsequent layers x_i. Consequently, the final output x_5 will be available during the same time interval the sensor is providing its output, this is, during interval 0 to 1 ms.

An immediate feature that the *pseudo-simultaneity* between input and output event flows allows, is the possibility of efficiently implementing feedback systems, as feedback would be instantaneous without any need to iterate for convergence. However, this feature is not exploited in present day ConvNets, because they are purely feed-forward.

3.5. FRAME-CONSTRAINED FIX-PIXEL-VALUE ConvNets

In this section we present a run-time programmable data-flow architecture, specially tailored for Frame-Constrained Fix-Pixel-Value ConvNets. We will refer to this implementation as the *FC-ConvNet Processor*. The processor receives sequences of still images (frames). For each frame, pixels have fix (constant) values. The architecture presented here has been fully coded in hardware description language (HDL) that target both ASIC synthesis and programmable hardware like FPGAs.

A schematic summary of the *FC-ConvNet Processor* system is presented in **Figure 7A**. The main components are: (1) a *Control Unit* (implemented on a general-purpose CPU), (2) a grid of independent *Processing Tiles (PTs)*, each containing a routing multiplexer (MUX) and local operators, and (3) a *Smart DMA* interfacing external memory via a standard controller.

The architecture presented here proposes a very different paradigm to parallelism, as each PT only contains *useful computing logic*. This allows us to use the silicon surface in a most efficient way. In fact, where a typical multi-processor system would be able to use 50 cores, the proposed data-flow grid could implement 500 tiles.

For image processing tasks (ConvNets in this case), the following observations/design choices fully justify the use of this type of grid:

- Throughput is a top priority. Indeed, most of the operations performed on images are replicated over both dimensions of images, usually bringing the amount of similar computations to a number that is much larger than the typical latencies of a pipelined processing tile.
- Reconfiguration time has to be low (in the order of the system's latency). This is achieved by the use of a common run-time configuration bus. Each module in the design has a set of configurable parameters, routes or settings (depicted as squares on **Figure 7A**), and possesses a unique address on the network. Groups of similar modules also share a broadcast address, which dramatically speeds up their reconfiguration.
- The processing elements in the grid should be as coarse grained as permitted, to maximize the ratio between computing logic and routing logic.
- The processing elements should not have any internal state, but should just passively process any incoming data. The task of sequencing operations is done by the global control unit, which stores the state and simply configures the entire grid for a given operation, lets the data-flow in, and prepares the following operation.

Figure 7B shows how the grid can be configured to compute a sub-part of a ConvNet (a sum of two convolutions is fed to a non-linear mapper). In that particular configuration, both the kernels and the images are streams loaded from external memory (the filter kernels can be pre-loaded in local caches concurrently to another operation). By efficiently alternating between grid reconfiguration and data streaming, an entire ConvNet can be computed (unrolled in time).

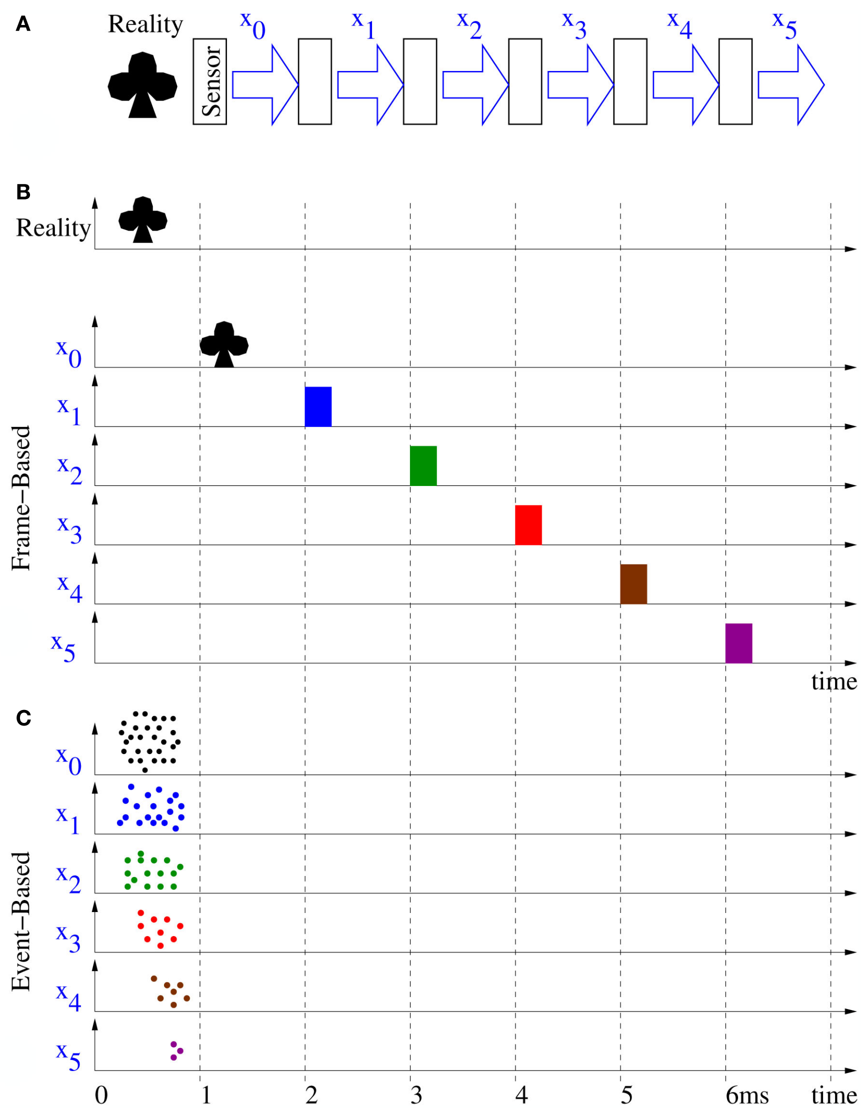


FIGURE 6 | Illustration of pseudo-simultaneity concept extrapolated to multiple layers. (A) Vision system composed of Vision Sensor and five sequential processing stages, like in a ConvNet. **(B)** Timing in a Frame-constraint system with 1 ms frame time for sensing and per stage processing. **(C)** Timing in an Event-driven system with micro-second delays for sensor and processor events.

A compiler takes a software representation of a trained ConvNet, and produces the binary code to be executed on the *Control Unit*. The ConvNet Processor can be reprogrammed with new binary code at run-time.

The compiler typically executes the following operations:

- Step 1: Analyses a given ConvNet layer by layer, and performs cross-layer optimizations (like layer combinations and merging).
- Step 2: Creates a memory map with efficient packing, to place all intermediate results (mostly feature maps for ConvNets) in a minimal memory footprint.
- Step 3: Decomposes each layer from step 1 into sequences of grid reconfigurations and data streams. Each reconfiguration results in a set of operations to be performed by the Control Unit and each data stream results in a set of operations for the Smart DMA (to read/write from/to external memory).
- Step 4: Results from Step 3 are turned into a fully sequential binary code for the Control Unit.

Our architecture was implemented on two FPGAs, a low-end Virtex 4 with limited memory bandwidth and a high-end Virtex-6 with fourfold memory bandwidth.

Figure 8 shows the time taken to compute a typical ConvNet trained for scene analysis/obstacle detection (pixel-wise classification, see Hadsell et al., 2009), on different computing platforms. The CPU implementation is classical C implementation using BLAS libraries. The GPU implementation is a hand-optimized implementation that uses as many of the cores as possible. The GPU, an nVidia 9400 M is a middle-range GPU optimized for

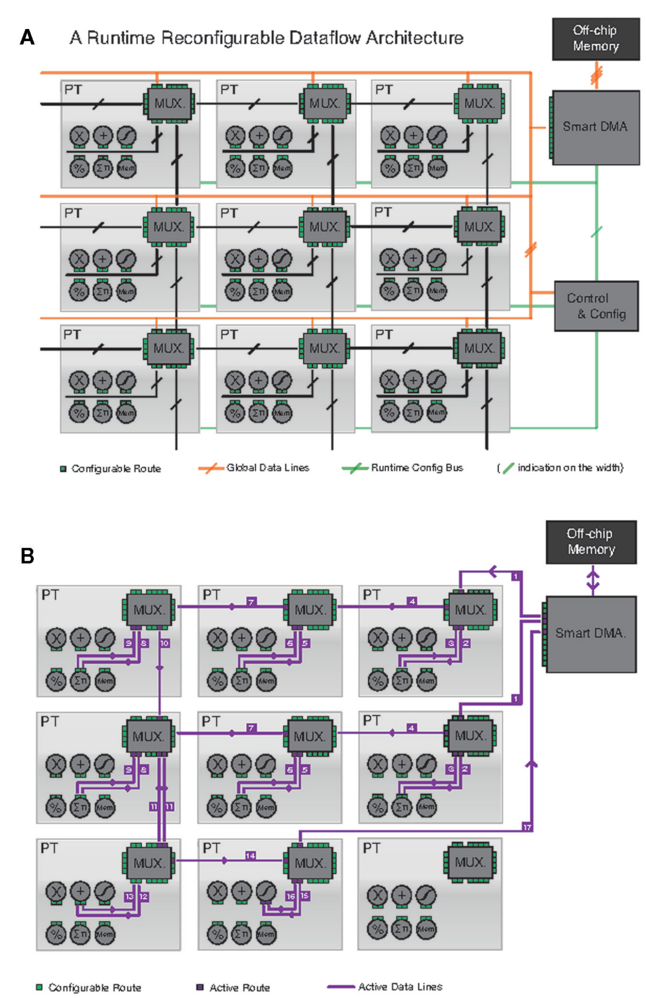


FIGURE 7 | (A) A data-flow computer. A set of run-time configurable processing tiles are connected on a 2D grid. They can exchange data with their 4 neighbors and with an off-chip memory via global lines. **(B)** The grid is configured for a more complex computation that involves several tiles: the 3 top tiles perform a 3 × 3 convolution, the 3 intermediate tiles another 3 × 3 convolution, the bottom left tile sums these two convolutions, and the bottom center tile applies a function to the result.

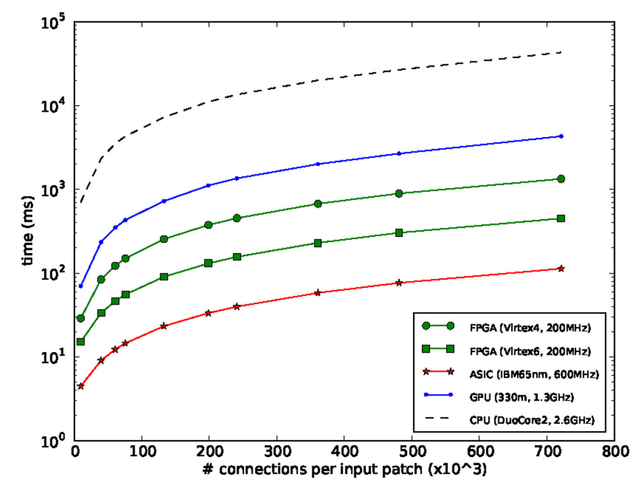


FIGURE 8 | Computing time for a typical ConvNet, versus the number of connections used for training the network.

low-power. As can be seen, the most generic hardware (CPU) is the least efficient because it is less parallel and relies on heavy processor-memory traffic. The GPU improves about an order of magnitude, as more parallelism is achieved. FPGA implementations can be made to exploit massive parallelism with high-bandwidth memories, thus achieving much higher efficiencies. Finally, a dedicated ASIC in a high-end technology would be optimum.

3.6. COMPARISON BETWEEN FRAME-CONSTRAINED AND FRAME-FREE SPIKING ConvNets

In order to compare Frame-Constrained vs. Frame-Free spiking hardware performance of ConvNets implementations, we need to be aware of the fundamental difference between information coding of both approaches.

In a Frame-Constrained vision system, visual reality is sampled at a rate T_{frame}. The input to the system is then, for each T_{frame}, an array of $N \times M$ pixels each carrying an n-bit value. There is a fixed amount of input information per frame. For a given ConvNet topology (as in **Figure 1**), one knows exactly the number and type of operations that have to be carried out starting from the input frame. Depending on the available hardware resources (multipliers, adders, accumulators, etc) one can estimate the delay in processing the full ConvNet for one input image, independently on the content of the image. If the full ConvNet operators can be mapped one by one onto respective hardware operators, then no intermediate computation data has to be fetched in and out from the chip/FPGA to external memory. This is the ideal case. However, in practical implementations to-date, either the input image is processed by patches, or the ConvNet is processed by parts within the hardware, or a combination of both, using extensive chip/FPGA to external memory traffic. Let's call R_{hw} the ratio between the available hardware resources and all the hardware resources a given ConvNet would require to compute the full input frame without fetching intermediate data to/from external memory. Then, in Frame-Constrained Fix-Pixel-Value ConvNets speed is a strong function of R_{hw} and the external memory bandwidth.

In a Frame-Free Spiking System, sensor pixels generate spikes continuously and asynchronously. Visual information is represented by a flow of events, each defined in 3D (x, y, t). Many times an event carries also "sign" information (positive or negative). The number of spikes per second in the visual flow is highly dependent on scene information content (as opposed to the Frame-Constrained case). In Frame-Free Spiking systems, the full ConvNet structure (as in **Figure 1**) must be available in hardware. Consequently, $R_{hw} = 1$. This is due to the fact that visual information at each node of the ConvNet is represented by a sequence or flow of events that "fill" the time scale and keep synchrony among all nodes. The great advantage of this is that the different flows are practically simultaneous because of the "pseudo-simultaneity" property of input-to-output flows in each ConvNet module. The processing delay between input-to-output flows is determined mainly by the statistics of the input event flow data. For example, how many space-time correlated input events need to be collected that represent a given shape. If one tries to time-multiplex the hardware resources (for implementing larger networks, for example) then the flows would need to be sampled and stored, which would convert the system into a Frame-Constrained one. Consequently, if one wants to scale up a Frame-Free Spiking ConvNet, then it is necessary to add more hardware modules. In principle, this should be simple, as inter-module links are asynchronous and modules are all alike. As the system scales up, however, processing speed is not degraded, as it is determined by the statistical information content of the input event flow. Note that this is a fundamental difference with respect to Frame-constrained systems, where one needs to first wait for the sensor to provide a full frame before starting processing it. Scaling up a spiking system does not affect the pseudo-simultaneity property. An important limitation will be given by the inter-module event communication bandwidth. Normally, event rate lowers as processing is performed at subsequent stages. Thus the highest event rate is usually found at the sensor output. Consequently, it is important that the sensors include some kind of pre-processing (such as spatial or temporal contrast) to guarantee a rather sparse event count.

Although present day ConvNets are purely feed-forward structures, it is widely accepted that computations in brains exploit extensive use of feedback between processing layers. In a Frame-constraint system, implementing feedback would require to iterate each feed-forward pass until convergence, for each frame. On the other hand, in Frame-free event-driven systems, since input and output flows at each module are instantaneous, feedback would be instantaneous as well, without any need for iterations.

Another big difference between Frame-Constrained and Frame-Free implementations is that the first one is technologically more mature while the second one is very incipient and in research phase.

Table 2 summarizes the main differences between both approaches in terms of how data is processed, whether hardware multiplexing is possible, how hardware can be scaled-up, and what determines processing speed and power consumption. Note that AER spiking hardware is easily expandable in a modular fashion by simply interconnecting AER links (Serrano-Gotarredona et al., 2009; Zamarreño-Ramos et al., under review). However, expanding the FPGA hardware described in Section 4 is not so straight forward and dedicated *ad hoc* techniques need to be developed.

Table 3 compares numerically performance figures of comparable ConvNets implemented using either Frame-Constrained fix-pixel-value or Frame-free spiking-dynamic-pixel techniques. The first two columns show performance figures of arrays of Gabor filters synthesized into Virtex-6 FPGAs. The Purdue/NYU system implements an array of 16 parallel 10×10 kernel Gabor filters operating on input images of 512×512 pixels with a delay of 5.2 ms, thus equivalent to 4 M-neurons with 400 M-synapses and a computing power of 7.8×10^{10} conn/s. The IMSE/US system implements an array of 64 Gabor filters operating on input visual scenes of 128×128 pixels with delays of 3 μs per-event per module, thus equivalent to 0.26 M-neurons with 32 M-synapses and a computing power of 2.6×10^9 conn/s.

Note that while the 5.2 ms delay of the Purdue/NYU Frame-Constraint system represents the filtering delay of 16 ConvModules, the 3-μs/event delay of the IMSE/US system does not represent a filtering delay. This number simply characterizes the intrinsic speed of the hardware. The filtering or recognition delay will be determined by the statistical time distribution of input events. As soon as enough input events are available that allow the system to provide a recognition decision, an output event will be produced (3 μs after the last input event).

Table 2 | Frame-free vs. frame-constrained.

	Frame-free	**Frame-constrained**
Data processing	Per-event, resulting in pseudo-simultaneity	Per frame/patch
Hardware multiplexing	Not possible	Possible
Hardware up-scaling	By adding modules	*Ad hoc*
Speed	Determined by statistics of input stimuli	Determined by number and type of operations, available hardware resources and their speed
Power consumption	Determined by module power per-event, and inter-module communication power per-event	Determined by power of processor(s) and memory fetching requirements
Feedback	Instantaneous. No need to iterate	Need to iterate until convergence for each frame

Table 3 | Performance comparison.

	Purdue/NYU	**IMSE/US**	**3D ASIC**	**Grid 40 nm**
Input scene size	521×512	128×128	512×512	512×512
Delay	5.2 ms/frame	3 μs/event	1.3 ms/frame	10 ns/events
Gabor array	16 convs 10×10 kernels	64 convs 11×11 kernels	16 convs 10×10 kernels	100 convs 32×32 kernels
Neurons	4.05×10^6	2.62×10^5	4.05×10^6	10^8
Synapses	4.05×10^8	3.20×10^7	4.05×10^8	10^{11}
Conn/s	7.8×10^{10}	2.6×10^9	3×10^{11}	4×10^{13}

The third and fourth columns represent performance estimations for futuristic Frame-constrained and Frame-free systems. Column 3 corresponds to the ASIC systems projected for a high-end 3D technology (see **Figure 7**), where speed is improved a factor four for a given number of connections with respect to the Virtex-6 realization. Column four corresponds to the estimated performance for an array of 100 reconfigurable multi-module 40 nm technology chips. Based on the performance figures of an already tested event-driven ConvChip fabricated in 0.35 μm CMOS (Camuñas-Mesa et al., 2011, 2012), which holds an array of 64×64 pixels in about 5 mm $\times$ 5 mm, it is reasonable to expect that a 1-cm^2 die fabricated in 40 nm CMOS could hold 1 million neurons with 1G-synapses. In order to improve event throughput, processing pixels should be tiled into slices to avoid very long lines and pipeline/parallelize event processing. Off-chip event communication should be done serially (Zamarreño-Ramos et al., 2011a,b), and possibly using multiple I/O ports to improve interchip throughput. All this could probably improve event throughput by a factor of 100 with respect to the presented prototype. Consequently, we might consider as viable, event throughputs in the order of 10^8 eps (events per second) per chip. Using AER-mesh techniques (Zamarreño-Ramos, 2011; Zamarreño-Ramos et al., under review) to assemble modularly a grid of 10×10 such chips on a (stackable) PCB would allow for a ConvNet system with about 10^8 neurons and 10^{11} synapses, which is about 1% of the human cerebral cortex (Azevedo et al., 2009), in terms of number of neurons and synapses. The brain is certainly more sophisticated and has other features not considered here, such as learning, synaptic complexity, stochastic, and molecular computations, and more.

In order to compare the effective performance capability of Frame-Constraint versus Frame-Free hardware, the most objective criteria is to compare their "connections/second" capability, as shown in the bottom of **Table 3**. However, these numbers should also not be judged as strictly equivalent, because while the Frame-Free version computes connections/sec on active pixels only, the Frame-Constraint version has to compute connection/s for all pixels thus introducing an extra overhead. This overhead depends on the statistical nature of the data.

4. CONCLUSION

We have presented a comparison analysis between Frame-Constrained and Frame-Free Implementations of ConvNet Systems for application in object recognition for vision. We have presented example implementations of Frame-Constrained FPGA realization of a full ConvNet system, and partial convolution processing stages (or combination of stages) using spiking AER convolution hardware using either VLSI convolution chips or FPGA realizations. The differences between the two approaches in terms of signal representations, computation speed, scalability, and hardware multiplexing have been established.

REFERENCES

Azadmehr, M., Abrahamsen, J., and Häfliger, P. (2005). "A foveated AER imager chip," in *Proceedings of the IEEE International Symposium on Circuits and Systems. (ISCAS)* (Kobe: IEEE Press), 2751–2754.

Azevedo, F. A., Carvalho, L. R., Grinberg, L. T., Farfel, J. M., Ferretti, R. E., Leite, R. E., Jacob Filho, W., Lent, R., and Herculano-Houzel, S. (2009). Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain. *J. Comp. Neurol.* 513, 532–541.

Boahen, K. (1999). "Retinomorphic chips that see quadruple images," in *Proceedings of the International Conference Microelectronics for Neural, Fuzzy and Bio-Inspired Systems (Microneuro)* (Granada: IEEE Press), 12–20.

Boser, B., Säckinger, E., Bromley, J., LeCun, Y., and Jackel, L. (1991). An analog neural network processor with programmable topology. *IEEE J. Solid State Circuits* 26, 2017–2025.

Camuñas-Mesa, L., Acosta-Jiménez, A., Zamarreño-Ramos, C., Serrano-Gotarredona, T., and Linares-Barranco, B. (2011). A convolution processor chip for address event vision sensors with 155ns event latency and 20Meps throughput. *IEEE Trans. Circuits Syst.* 58, 777–790.

Camuñas-Mesa, L., Zamarreño-Ramos, C., Linares-Barranco, A., Acosta-Jiménez, A., Serrano-Gotarredona, T., and Linares-Barranco, B. (2012). An event-driven convolution processor module for event-driven vision sensors. *IEEE J. Solid State Circuits* 47, 504–517.

Chan, V., Liu, S.-C., and van Schaik, A. (2007). AER EAR: a matched silicon cochlea pair with address event representation interface. *IEEE Trans. Circuits Syst. Part I* 54, 48–59.

Chen, S., and Bermak, A. (2007). Arbitrated time-to-first spike CMOS image sensor with on-chip histogram equalization. *IEEE Trans. VLSI Syst.* 15, 346–357.

Chicca, E., Whatley, A. M., Lichtsteiner, P., Dante, V., Delbrück, T., Del Giudice, P., Douglas, R. J., and Indiveri, G. (2007). A multichip pulse-based neuromorphic infrastructure and its application to a model of orientation selectivity. *IEEE Trans. Circuits Syst. Part I* 54, 981–993.

Choi, T. Y. W., Merolla, P., Arthur, J., Boahen, K., and Shi, B. E. (2005). Neuromorphic implementation of orientation hypercolumns. *IEEE Trans. Circuits Syst. Part I* 52, 1049–1060.

Cloutier, J., Cosatto, E., Pigeon, S., Boyer, F., and Simard, P. Y. (1996). "Vip: an fpga-based processor for image processing and neural networks," in *Proceedings of the Fifth International Conference on Microelectronics for Neural Networks and Fuzzy Systems MicroNeuro'96* (Lausanne: IEEE Press), 330–336.

Costas-Santos, J., Serrano-Gotarredona, T., Serrano-Gotarredona, R., and Linares-Barranco, B. (2007). A contrast retina with on-chip calibration for neuromorphic spike-based AER vision systems. *IEEE Trans. Circuits Syst. I Reg. Papers* 54, 1444–1458.

Culurciello, E., Etienne-Cummings, R., and Boahen, K. (2003). A biomorphic digital image sensor. *IEEE J. Solid State Circuits* 38, 281–294.

Delbrück, T. (2005). http://jaer.wiki.sourceforge.net

Delbrück, T. (2008). "Frame-free dynamic digital vision," in *Proceedings of International Symposium on Secure-Life Electronics, Advanced Electronics for Quality Life and Society* (Tokyo: University of Tokyo), 21–26.

Hadsell, R., Sermanet, P., Erkan, A., Ben, J., Han, J., Flepp, B., Muller, U., and LeCun, Y. (2007). 'On-line learning for offroad robots: using spatial label propagation to learn long-range traversability," in *Proceedings of Robotics Science and Systems '07*, MIT Press, Cambridge.

Hadsell, R., Sermanet, P., Scoffier, M., Erkan, A., Kavackuoglu, K., Muller, U., and LeCun, Y. (2009). Learning long-range vision for autonomous off-road driving. *J. Field Robotics* 26, 120–144.

Huang, F.-J., and LeCun, Y. (2006). "Large-scale learning with svm and convolutional nets for generic object categorization," in *Proceedings of Computer Vision and Pattern Recognition Conference (CVPR'06)* (New York: IEEE Press).

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

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1990). "Handwritten digit recognition with a back-propagation network," *In NIPS'89*, MIT Press, Denver.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998a). Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86, 2278–2324.

LeCun, Y., Bottou, L., Orr, G., and Muller, K. (1998b). "Efficient backprop," in *Neural Networks: Tricks of the Trade*, eds G. Orr, and K. Muller (Springer).

Leñero-Bardallo, J. A., Serrano-Gotarredona, T., and Linares-Barranco, B. (2010). A five-decade dynamic-range ambient-light-independent calibrated signed-spatial-contrast AER retina with 0.1ms latency and optional time-to-first-spike mode. *IEEE Trans. Circuits Syst. I Reg. Papers* 57, 2632–2643.

Leñero-Bardallo, J. A., Serrano-Gotarredona, T., and Linares-Barranco, B. (2011). A 3.6μs latency asynchronous frame-free event-based dynamic vision sensor. *IEEE J. Solid State Circuits* 46, 1443–1455.

Lichtsteiner, P., Posch, C., and Delbrück, T. (1998). A 128Ã – 128 120db 15us latency asynchronous temporal contrast vision sensor. *IEEE J. Solid State Circuits* 43, 566–576.

Lyu, S., and Simoncelli, E. P. (2008). "Nonlinear image representation using divisive normalization," in *Computer Vision and Pattern Recognition*, IEEE, Anchorage.

Massari, N., Gottardi, M., Jawed, S. A., and Soncini, G. (2008). A 100uw 64×128-pixel contrast-based asynchronous binary vision sensor for wireless sensor networks. *IEEE ISSCC Dig. Tech. Papers* 588–638.

Neubauer, C. (1998). Evaluation of convolution neural networks for visual recognition. *IEEE Trans. Neural Netw.* 9, 685–696.

Osadchy, R., Miller, M., and LeCun, Y. (2005). "Synergistic face detection and pose estimation with energy-based model," in *Advances in Neural Information Processing Systems (NIPS 2004)* (Vancouver: MIT Press).

Oster, M., Yingxue, W., Douglas, R., and Shih-Chii, L. (2008). Quantification of a spike-based winner-take-all vlsi network. *IEEE Trans. Circuits. Syst. Part* 1 55, 3160–3169.

Pérez-Carrasco, J. A. (2011). *A Simulation Tool for Building and Analyzing Complex and Hierarchically Structured AER Visual Processing Systems.* Ph.D. thesis, IMSE-CNM-CSIC, Universidad de Sevilla, Sevilla.

Pinto, N., Cox, D. D., and DiCarlo, J. J. (2008). Why is real-world visual object recognition hard? *PLoS Comput. Biol.* 4, e27. doi:10.1371/journal.pcbi

Posch, C., Matolin, D., and Wohlgenannt, R. (2010). "A QVGA 143dB DR asynchronous address-event PWM dynamic vision and image sensor with lossless pixel-level video compression and time-domain CDS," in *ISSCC Digest of Technical Papers*, San Francisco, in press.

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

Ruedi, P. F., Heim, P., Gyger, S., Kaess, F., Arm, C., Caseiro, R., Nagel, J.-L., and Todeschini, S. (2009). "An soc combining a 132db qvga pixel array and a 32b dsp/mcu processor for vision applications," in *IEEE ISSCC Digest of Technical Papers*, San Francisco, 46–47, 47a.

Ruedi, P. F., Heim, P., Kaess, F., Grenet, E., Heitger, F., Burgi, P.-Y., Gyger, S., and Nussbaum, P. (2003). A 128×128, pixel 120-db dynamic-range vision-sensor chip for image contrast and orientation extraction. *IEEE J. Solid State Circuits* 38, 2325–2333.

Säckinger, E., Boser, B., Bromley, J., LeCun, Y., and Jackel, L. D. (1992). Application of the ANNA neural network chip to high-speed character recognition. *IEEE Trans. Neural Netw.* 3, 498–505.

Serrano-Gotarredona, R., Oster, M., Lichtsteiner, P., Linares-Barranco, A., Paz-Vicente, R., Gómez-Rodríguez, F., Camuñas-Mesa, L., Berner, R., Rivas-Pérez, M., Delbrück, T., Liu, S.-C., Douglas, R., Häfliger, P., Jiménez-Moreno, G., Ballcels, A. C., Serrano-Gotarredona, T., Acosta-Jiménez, A. J., and Linares-Barranco, B. (2009). CAVIAR: a 45k neuron, 5M synapse, 12G connects/s AER hardware sensory-processing-learning-actuating system for high-speed visual object recognition and tracking. *IEEE Trans. Neural Netw.* 20, 1417–1438.

Serrano-Gotarredona, R., Serrano-Gotarredona, T., Acosta-Jiménez, A., and Linares-Barranco, B. (2006). A neuromorphic cortical-layer microchip for spike-based event processing vision systems. *IEEE Trans. Circuits Syst. I Regul. Papers* 53, 2548–2566.

Serrano-Gotarredona, R., Serrano-Gotarredona, T., Acosta-Jiménez, A., Serrano-Gotarredona, C., Pérez-Carrasco, J. A., Linares-Barranco, B., Linares-Barranco, A., Jiménez-Moreno, G., and Civit-Ballcels, A. (2008). On real-time AER 2-D convolution hardware for neuromorphic spike-based cortical processing. *IEEE Trans. Neural Netw.* 19, 1196–1219.

Serrano-Gotarredona, T., Andreou, A. G., and Linares-Barranco, B. (1999). AER image filtering architecture for vision processing systems. *IEEE Trans. Circuits Syst. Part I Fundam. Theory Appl.* 46, 1064–1071.

Serre, T. (2006). *Learning a Dictionary of Shape-Components in Visual Cortex: Comparison with Neurons, Humans and Machines.* Ph.D. thesis, MIT, Boston.

Shepherd, G. (1990). *The Synaptic Organization of the Brain*, 3rd Edn. Oxford: Oxford University Press.

Sivilotti, M. A. (1991). "Wiring considerations in analog VLSI systems, with application to field-programmable networks," in *Technical Report*, California Institute of Technology, Pasadena.

Teixeira, T., Culurciello, E., and Andreou, A.G. (2006). "An address-event image sensor network," in *IEEE International Symposium on Circuits and Systems, ISCAS '06* (Kos: IEEE), 4467–4470.

Thorpe, S., Fize, D., and Marlot, C. (1996). Speed of processing in the human visual system. *Nature* 381, 520–522.

Vernier, P., Mortara, A., Arreguit, X., and Vittoz, E. A. (1997). An integrated cortical layer for orientation enhancement. *IEEE J. Solid State Circuits* 32, 177–186.

Vogelstein, R. J., Mallik, U., Culurciello, E., Cauwenberghs, G., and Etienne-Cummings, R. (2007). A multichip neuromorphic system for spike-based visual information processing. *Neural Comput.* 19, 2281–2300.

Zaghloul, K. A., and Boahen, K. (2004). Optic nerve signals in a neuromorphic chip: parts 1 and 2. *IEEE Trans.Biomed. Eng.* 51, 657–675.

Zamarreño-Ramos, C. (2011). *Towards Modular and Scalable High-Speed AER Vision Systems.* Ph.D. thesis, IMSE-CNM-CSIC, Universidad de Sevilla, Sevilla.

Zamarreño-Ramos, C., Serrano-Gotarredona, T., Linares-Barranco, B., Kulkarni, R., and Silva-Martinez, J. (2011a). "Voltage mode driver for low power transmission of high speed serial aer links," in *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS 2011)* (Rio de Janeiro), 2433–2436.

Zamarreño-Ramos, C., Serrano-Gotarredona, T., and Linares-Barranco, B. (2011b). An instant-startup jitter-tolerant manchester-encoding serializer/deserializar scheme for event-driven bit-serial lvds inter-chip aer links. *IEEE Trans. Circuits Syst. Part I* 58, 2647–2660.

Conflict of Interest Statement: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Received: 28 October 2011; accepted: 21 February 2012; published online: 12 April 2012.

Citation: Farabet C, Paz R, Pérez-Carrasco J, Zamarreño-Ramos C, Linares-Barranco A, LeCun Y, Culurciello E, Serrano-Gotarredona T and Linares-Barranco B (2012) Comparison between frame-constrained fix-pixel-value and frame-free spiking-dynamic-pixel convNets for visual processing. Front. Neurosci. ***6****:32. doi: 10.3389/fnins.2012.00032*

This article was submitted to Frontiers in Neuromorphic Engineering, a specialty of Frontiers in Neuroscience.