YodaNN: An Architecture for Ultralow Power Binary-Weight CNN Acceleration

Renzo Andri, Lukas Cavigelli, Davide Rossi, and Luca Benini, *Fellow, IEEE*

***Abstract*—Convolutional neural networks (CNNs) have revolutionized the world of computer vision over the last few years, pushing image classification beyond human accuracy. The computational effort of today's CNNs requires power-hungry parallel processors or GP-GPUs. Recent developments in CNN accelerators for system-on-chip integration have reduced energy consumption significantly. Unfortunately, even these highly optimized devices are above the power envelope imposed by mobile and deeply embedded applications and face hard limitations caused by CNN weight I/O and storage. This prevents the adoption of CNNs in future ultralow power Internet of Things end-nodes for near-sensor analytics. Recent algorithmic and theoretical advancements enable competitive classification accuracy even when limiting CNNs to binary (+1/−1) weights during training. These new findings bring major optimization opportunities in the arithmetic core by removing the need for expensive multiplications, as well as reducing I/O bandwidth and storage. In this paper, we present an accelerator optimized for binary-weight CNNs that achieves 1.5 TOp/s at 1.2 V on a core area of only 1.33 million gate equivalent (MGE) or 1.9 mm^2 and with a power dissipation of 895 μW in UMC 65-nm technology at 0.6 V. Our accelerator significantly outperforms the state-of-the-art in terms of energy and area efficiency achieving 61.2 TOp/s/W@0.6 V and 1.1 TOp/s/MGE@1.2 V, respectively.**

***Index Terms*—ASIC, binary weights, convolutional neural networks (CNNs), hardware accelerator, Internet of Things (IoT).**

I. Introduction

CONVOLUTIONAL neural networks (CNNs) have been achieving outstanding results in several complex tasks such as image recognition [2]–[4], face detection [5], speech recognition [6], text understanding [7], [8], and artificial intelligence in games [9], [10]. Although optimized software implementations have been largely deployed on mainstream systems [11], CPUs [12], and GPUs [13] to deal with several state of the art CNNs, these platforms are obviously not able to fulfill the power constraints imposed by mobile and Internet of Things (IoT) end-node devices. On the other hand, sourcing out all CNN computation from IoT end-nodes to data servers is extremely challenging and power consuming, due to the large communication bandwidth required to transmit the data streams. This prompts for the need of specialized architectures to achieve higher performance at lower power within the end-nodes of the IoT.

A few research groups exploited the customization paradigm by designing highly specialized hardware to enable CNN computation in the domain of embedded applications. Several approaches leverage field-programmable gate arrays (FPGAs) to maintain post-fabrication programmability, while providing significant boost in terms of performance and energy efficiency [14]. However, FPGAs are still two orders of magnitude less energy-efficient than ASICs [15]. Moreover, CNNs are based on a very reduced set of computational kernels (i.e., convolution, activation, and pooling), but they can be used to cover several application domains (e.g., audio, video, and biosignals) by simply changing weights and network topology, relaxing the issues with nonrecurring engineering which are typical in ASIC design.

Among CNN ASIC implementations, the precision of arithmetic operands plays a crucial role in energy efficiency. Several reduced-precision implementations have been proposed recently, relying on 16-bit, 12-bit, or 10-bit of accuracy for both operands and weights [15]–[19], exploiting the intrinsic resiliency of CNNs to quantization and approximation [20], [21]. In this paper, we take a significant step forward in energy efficiency by exploiting recent research on binary-weight CNNs [22], [23]. BinaryConnect is a method which trains a deep neural network with binary weights during the forward and backward propagation, while retaining the precision of the stored weights for gradient descent optimization. This approach has the potential to bring great benefits to CNN hardware implementation by enabling the replacement of multipliers with much simpler complement operations and multiplexers, and by drastically reducing weight storage requirements. Interestingly, binary-weight networks lead to only small accuracy losses on several well-known CNN benchmarks [24], [25].

In this paper, we introduce the first optimized hardware design implementing a flexible, energy-efficient and performance scalable convolutional accelerator supporting binary-weight CNNs. We demonstrate that this approach improves the energy efficiency of the digital core of the accelerator by

Manuscript received September 14, 2016; revised December 30, 2016 and February 20, 2017; accepted February 26, 2017. Date of publication March 14, 2017; date of current version December 20, 2017. This work was supported in part by the Swiss National Science Foundation under Grant 162524 (MicroLearn: Micropower Deep Learning), in part by the Armasuisse Science and Technology, and in part by the ERC MultiTherman Project under Grant ERC-AdG-291125. This paper was recommended by Associate Editor X. Li.

R. Andri and L. Cavigelli are with the Integrated Systems Laboratory, ETH Zurich, 8092 Zürich, Switzerland (e-mail: renzo.andri@iis.ee.ethz.ch).

D. Rossi is with the Department of Electrical, Electronic and Information Engineering, University of Bologna, 40136 Bologna, Italy.

L. Benini is with the Integrated Systems Laboratory, ETH Zurich, 8092 Zürich, Switzerland, and also with the Department of Electrical, Electronic and Information Engineering, University of Bologna, 40136 Bologna, Italy.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2017.2682138

5.1×, and the throughput by 1.3×, with respect to a baseline architecture based on 12-bit MAC units operating at a nominal supply voltage of 1.2 V. To extend the performance scalability of the device, we implement a latch-based standard cell memory (SCM) architecture for on-chip data storage. Although SCMs are more expensive than SRAMs in terms of area, they provide better voltage scalability and energy efficiency [26], extending the operating range of the device in the low-voltage region. This further improves the energy efficiency of the engine by 6× at 0.6 V, with respect to the nominal operating voltage of 1.2 V, and leads to an improvement in energy efficiency by 11.6× with respect to a fixed-point implementation with SRAMs at its best energy point of 0.8 V. To improve the flexibility of the convolutional engine we implement support for several kernel sizes ($1 \times 1 - 7 \times 7$), and support for per-channel scaling and biasing, making it suitable for implementing a large variety of CNNs. The proposed accelerator surpasses state-of-the-art CNN accelerators by 2.7× in peak performance with 1.5 TOp/s [27], by 10× in peak area efficiency with 1.1 TOp/s/MGE [28] and by 32× peak energy efficiency with 61.2 TOp/s/W [28].

II. Related Work

CNNs are reaching record-breaking accuracy in image recognition on small data sets like MNIST, SVHN, and CIFAR-10 with accuracy rates of 99.79%, 98.31%, and 96.53% [29]–[31]. Recent CNN architectures also perform very well for large and complex data sets such as ImageNet: GoogLeNet reached 93.33% and ResNet achieved a higher recognition rate (96.43%) than humans (94.9%). As the trend goes to deeper CNNs (e.g., ResNet uses from 18 up to 1001 layers, VGG OxfordNet uses 19 layers [32]), both memory and computational complexity increases. Although CNN-based classification is not problematic when running on mainstream processors or large GPU clusters with kW-level power budgets, IoT edge-node applications have much tighter, mW-level power budgets. This "CNN power wall" led to the development of many approaches to improve CNN energy efficiency, both at the algorithmic and at the hardware level.

A. Algorithmic Approaches

Several approaches reduce the arithmetic complexity of CNNs by using fixed-point operations and minimizing the word widths. Software frameworks, such as Ristretto focus on CNN quantization after training. For LeNet and Cifar-10 the additional error introduced by this quantization is less than 0.3% and 2%, respectively, even when the word width has been constrained to 4-bit [21]. It was shown that state-of-the-art results can be achieved quantizing the weights and activations of each layer separately [33], while lowering precision down to 2-bit (−1, 0, +1) and increasing the network size [20]. Moons *et al.* [34] analyzed the accuracy-energy tradeoff by exploiting quantization and precision scaling. Considering the sparsity in deeper layers because of the ReLU activation function, they detect multiplications with zeros and skip them, reducing run time and saving energy. They reduce power by 30× (compared to 16-bit fixed-point) without accuracy loss, or 225× with a 1% increase in error by quantizing layers independently.

BinaryConnect [25] proposes to binarize (−1, +1) the weights w_{fp}. During training, the weights are stored and updated in full precision, but binarized for forward and backward propagation. The following formula shows the deterministic and stochastic binarization function, where a "hard sigmoid" function σ is used to determine the probability distribution:

$$w_{b,\text{det}} = \begin{cases} 1, & \text{if } w_{fp} < 0 \\ -1, & \text{if } w_{fp} > 0 \end{cases}, \quad w_{b,\text{sto}} = \begin{cases} 1, & p = \sigma(w_{fp}) \\ -1, & p = 1 - \sigma \end{cases}$$

$$\sigma(x) = \text{clip}\left(\frac{x+1}{2}, 0, 1\right) = \max\left(0, \min\left(1, \frac{x+1}{2}\right)\right).$$

In a follow-up work [25], the same authors propose to quantize the inputs of the layers in the backward propagation to 3 or 4 bits, and to replace the multiplications with shift-add operations. The resulting CNN outperforms in terms of accuracy even the full-precision network. This can be attributed to the regularization effect caused by restricting the number of possible values of the weights.

Following this trend, Courbariaux and Bengio [24] and Rastegari *et al.* [23] considered also the binarization of the layer inputs, such that the proposed algorithms can be implemented using only XNOR operations. In these works, two approaches are presented.

1) Binary-weight-networks BWN which scale the output channels by the mean of the real-valued weights. With this approach they reach similar accuracy in the ImageNet data set when using AlexNet [2].
2) XNOR-Networks where they also binarize the input images. This approach achieves an accuracy of 69.2% in the top-five measure, compared to the 80.2% of the setup 1). Based on this paper, Wu [35] improved the accuracy up to 81% using log-loss with soft-max pooling, and he was able to outperform even the accuracy results of AlexNet. However, the XNOR-based approach is not mature enough since it has only been proven on a few networks by a small research community.

Similarly to the scaling by the batch normalization, Merolla *et al.* [36] evaluated different weight projection functions where the accuracy could even be improved from 89% to 92% on Cifar-10 when binarizing weights and scaling every output channel by the maximum-absolute value of all contained filters. In this paper, we focus on implementing a CNN inference accelerator for neural networks supporting per-channel scaling and biasing, and implementing binary weights and fixed-point activation. Exploiting this approach, the reduction of complexity is promising in terms of energy and speed, while near state-of-the-art classification accuracy can be achieved with appropriately trained binary networks [22], [23].

B. CNN Acceleration Hardware

There are several approaches to perform CNN computations on GPUs, which are able to reach a throughput up to 6 TOp/s

with a power consumption of 250 W [13], [37]. On the other hand, there is a clear demand for low-power CNN acceleration. For example, Google exploits in its data-centers a custom-made neural network accelerator called *Tensor Processing Unit* tailored to their TensorFlow framework. Google claims that they were able to reduce power by roughly 10× with respect to GP-GPUs [38]. Specialized functional units are also beneficial for low-power programmable accelerators which recently entered the market. A known example is the Movidius Myriad 2 which computes 100 GFLOPS and needs just 500 mW@600 MHz [39]. However, these low-power architectures are still significantly above the energy budget of IoT end-nodes. Therefore, several dedicated hardware architectures have been proposed to improve the energy efficiency while preserving performance, at the cost of flexibility.

Several CNN systems were presented implementing activation layer (mainly ReLU) and pooling (i.e., max pooling) [27], [28], [40]. In this paper, we focus on the convolution layer as this contributes most to the computational complexity [13]. Since convolutions typically rely on recent data for the majority of computations, sliding window schemes are typically used [17], [18], [40], [41] (e.g., in case of 7×7 kernels, 6×7 pixels are reused in the subsequent step). In this paper, we go even further and cache the values, such that we can reuse them when we switch from one to the next tile. In this way, only one pixel per cycle has to be loaded from the off-chip storage.

As the filter kernel sizes change from problem to problem, several approaches were proposed to support more than one fixed kernel size. Zero-padding is one possibility: in Neuflow the filter kernel was fixed to 9×9 and it was filled with zeros for smaller filters [42]. However, this means that for smaller filters unnecessary data has to be loaded, and that the unused hardware cannot be switched off. Another approach was presented by Chen *et al.* [41], who have proposed an accelerator containing an array of 14×12 configurable processing elements connected through a network-on-chip. The PEs can be adjusted for several filter sizes. For small filter sizes, they can be used to calculate several output channels in parallel or they can be switched-off. Even though this approach brings flexibility, all data packets have to be labeled, such that the data can be reassembled in a later step. Hence, this system requires a lot of additional multiplexers and control logic, forming a bottleneck for energy efficiency. To improve the flexibility of YodaNN[1] we propose an architecture that implements several kernel sizes (1×1, 2×2, ..., 7×7). Our hardware exploits a native hardware implementation for 7×7, 5×5, and 3×3 filters, in conjunction with zero-padding to implement the other kernel sizes.

Another approach minimizes the on-chip computational complexity exploiting the fact that due to the ReLU activation layer, zero-values appear quite often in CNNs. In this way some of the multiplications can be bypassed by means of zero-skipping [41]. This approach is also exploited by Reagen *et al*. [43] and Albericio *et al*. [44]. Another approach exploits that the weights' distribution shows a clear maximum around zero. Jaehyeong *et al.* [40] proposed in their work a small 16-bit multiplier, which triggers a stall and calculation of the higher-order bits only when an overflow is detected, which gives an improvement of 56% in energy efficiency. The complexity can be reduced further by implementing quantization scaling as described in Section II-A. Even though most approaches work with fixed-point operations, the number of quantization bits is still kept at 24-bit [28], [40] or 16-bit [17], [18], [27], [42], [45].

To improve throughput and energy efficiency, Han *et al*. [46] presented compressed deep neural networks, where the number of different weights are limited, and instead of saving or transmitting full precision weights, the related indices are used. They presented a neural networks accelerator, called efficient inference engine (EIE), exploiting network pruning and weight sharing (deep compression). For a network with a sparsity as high as 97%, EIE reaches an energy efficiency of 5 TOp/s/W, and a throughput of 100 GOp/s, which is equal to a throughput of 3 TOp/W for the equivalent noncompressed network [47]. Even though this outperforms the previous state-of-the-art by 5×, we can still demonstrate a 12× more efficient design exploiting binary weights. Jaehyeong *et al.* [40] used PCA to reduce the dimension of the kernels. Indeed, they showed that there is a strong correlation among the kernels, which can be exploited to reduce their dimensionality without major influence on accuracy. This actually reduces the energy needed to load the chip with the filters and reduces the area to save the weights, since only a small number of bases and a reduced number of weight components need to be transmitted. On the other hand, it also increases the core power consumption, since the weights have to be reconstructed on-the-fly. With binary weights, we were able to reduce the total kernel data by 12×, which is similar to the 12.5× reported in [40]. On the other hand, YodaNN outperforms their architecture by 43× in terms of energy efficiency thanks to its simpler internal architecture that do not require on-the-fly reconstruction. Some CNN accelerators have been presented exploiting analog computation: in one approach [48], part of the computation was performed partially on the camera sensor chip before transmitting the data to the digital processing chip. Another mixed-signal approach [50] looked into embedding part of the CNN computation in a memristive crossbar. Efficiencies of 960 GOp/s [48] and 380 GOp/s/W [49] were achieved. YodaNN outperforms these approaches by 64× and 161×, respectively, thanks to aggressive discretization and low-voltage digital logic.

The next step consists in quantizing the weights to a binary value. However, this approach has only been implemented on Nvidia GTX750 GPU leading to a 7× run-time reduction [24]. In this paper, we present the first hardware accelerator optimized for binary weights CNN, fully exploiting the benefits of the reduction in computational complexity boosting area and energy efficiency. Furthermore, the proposed design scales to deep near-threshold thanks to SCM and an optimized implementation flow, outperforming the state of the art by 2.7× in performance, 10× in area efficiency, and 32× in energy efficiency.

[1] YodaNN named after the Jedi master known from StarWars—"small in size but wise and powerful" [1].

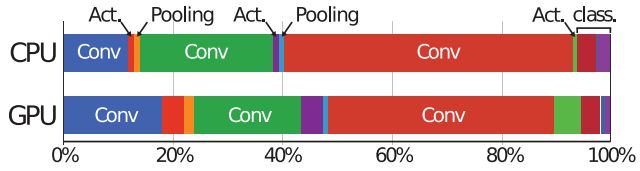


Fig. 1. Overview of execution time in a convolution neural network for scene labeling executed on CPU and GPU [13].

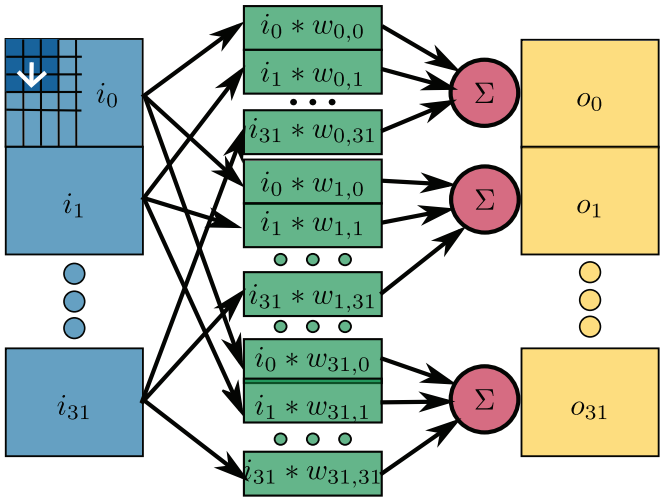


Fig. 2. 32×32 CNN layer, with input channels i_n and output channels o_k.

III. Architecture

A CNN consists of several layers, usually they are convolution, activation, pooling or batch normalization layers. In this paper, we focus on the convolution layers as they make up for the largest share of the total computation time. As can be seen in [13, Fig. 1], convolution layers make up for the largest fraction of compute time in CPU and GPU implementations. This is why we focus on convolution layers in this paper. A general convolution layer is drawn in Fig. 2 and it is described by (1), shown at the bottom of this page. A layer consists of n_{in} input channels, n_{out} output channels, and $n_{\text{in}} \cdot n_{\text{out}}$ kernels with $h_k \times b_k$ weights; we denote the matrix of filter weights as $w_{k,n}$. For each output channel k every input channel n is convolved with a different kernel $w_{k,n}$, resulting in the terms $\tilde{o}_{k,n}$, which are accumulated to the final output channel o_k. We propose a hardware architecture able to calculate $n_{\text{ch}} \times n_{\text{ch}}$ channels in parallel. If the number of input channels n_{in} is greater than n_{ch}, the system has to process the network $\lceil n_{\text{in}}/n_{\text{ch}} \rceil$ times and the results are accumulated off-chip. This adds only $\lceil n_{\text{in}}/n_{\text{ch}} \rceil - 1$ operations per pixel. In the following, we fix, for ease of illustration, the number of output channels to $n_{\text{ch}} = 32$ and the filter kernel size to $h_k = b_k = 7$. The system is composed of the following units (an overview can be seen in Fig. 3).

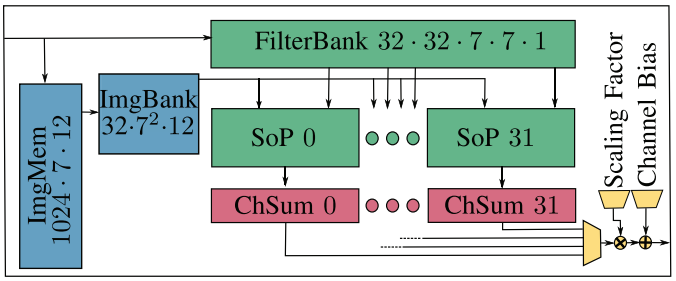


Fig. 3. General overview of the system with the image memory and image bank in blue, filter bank and SoP units in green, channel summer in red and the interleaved per-channel scaling, biasing and streaming-out units in yellow.

1) The *filter bank* is a shift register which contains the binary filter weights $w_{k,n}$ for the output channels $k \in \mathbb{N}_{<32}$ and input channels $n \in \mathbb{N}_{<32}$ ($n_{\text{in}} \cdot n_{\text{out}} \cdot h_k^2 \cdot 1\,\text{bit} = 6.4\,\text{kB}$) and supports column-wise left circular shift per kernel.
2) The *image memory* saves an image stripe of $b_k = 7$ width and 1024 height (10.8 kB), which can be used to cache $1024/n_{\text{in}} = 1024/32 = 32$ rows per input channel.
3) The *image bank* (ImgBank) caches a spatial window of $h_k \times b_k = 7 \times 7$ per input channel n (2.4 kB), which are applied to the sum-of-product (SoP) units. This unit is used to reduce memory accesses, as the $h_k - 1 = 6$ last rows can be reused when we proceed in a column-wise order through the input images. Only the lowest row has to be loaded from the image memory and the upper rows are shifted one row up.
4) *SoP Units (32, 1 Per Output Channel):* For every output channel k, the SoP unit k calculates the sum terms $\tilde{o}_{k,n}$, where in each cycle the contribution of a new input channel n is calculated.
5) *Channel Summer (ChSum) Units (32, 1 Per Output Channel):* The ChSum k accumulates the sum terms $\tilde{o}_{k,n}$ for all input channels n.
6) *1 Scale-Bias Unit:* After all the contributions of the input channels are summed together by the channel summers, this unit starts to scale and bias the output channels in an interleaved manner and streams them out.
7) *I/O Interface:* Manages the 12-bit input stream (input channels) and the two 12-bit output streams (output channels) with a protocol based on a blocking ready-valid handshaking.

A. Dataflow

The pseudo-code in Algorithm 1 gives an overview of the main steps required for the processing of convolution layers,

$$\mathbf{o_k} = \mathbf{C_k} + \sum_{n \in I} \underbrace{\mathbf{i_n} * \mathbf{w_{k,n}}}_{\tilde{\mathbf{o}}_{k,n}}, \qquad o_k(x,y) = C_k + \sum_{n \in I} \underbrace{\left(\sum_{a=0}^{b_k-1} \sum_{b=0}^{h_k-1} i_n(x+a, y+b) \cdot w_{k,n}(a,b) \right)}_{\tilde{o}_{k,n}(x,y)} \tag{1}$$

Algorithm 1 Dataflow Pseudo-Code

Require: weights $w_{k,n}$, input feature map $i_k(x, y)$
Ensure: $o_n = \sum_k i_k * w_{k,n}$
1: **for all** $y_{block} \in \{1, .., \lceil h_{\text{im}}/h_{max} \rceil\}$ **do**
2: **for all** $c_{out,block} \in \{1, .., \lceil n_{out}/n_{ch} \rceil\}$ **do**
3: **for all** $c_{in,block} \in \{1, .., \lceil n_{in}/n_{ch} \rceil\}$ **do**
4: – *YodaNN chip block*
5: Load Filters $w_{k,n}$
6: Load m colums, where $m = \begin{cases} h_k - 1, & \text{if not zero-padded} \\ \lfloor \frac{h_k - 1}{2} \rfloor, & \text{if zero-padded} \end{cases}$
7: Load m pixels of the $(m+1)^{\text{th}}$ column.
8:
9: – Parallel block 1
10: **for all** x **do**
11: **for all** y **do**
12: $\tilde{o}(c_{out} := \cdot, x, y) := 0$
13: **for all** c_{in} **do**
14:
15: ***– Single cycle block***
16: **for all** c_{out} **do**
17: **for all** (a,b) $\in \{-\lfloor \frac{h_k}{2} \rceil \leq a, b \leq \lceil \frac{h_k}{2} \rfloor\}$ **do**
18: $\tilde{o}_{c_{out}}(x, y) = \tilde{o}_{c_{out}}(x, y) + i_{c_{in}}(x+a, y+b) \cdot w_{c_{out},c_{in}}(a, b)$
19: **end for**
20: **end for**
21: **end for**
22: **end for**
23: **end for**
24: – Parallel block 2
25: **for all** x **do**
26: **wait until** $\tilde{o}_0(x, 0)$ is computed
27: **for all** y **do**
28: **for all** c_{out} **do**
29: ***– Single cycle block***
30: $o_{c_{out}}(x, y) = \alpha_{c_{out}} \tilde{o}_{c_{out}}(x, y) + \beta_{c_{out}}$
31: **output** $o_{c_{out}}(x, y)$
32: **end for**
33: **end for**
34: **end for**
35: **end for**
36: – Sum the input channel blocks:
37: $o_{n,final} = \sum_{c_{in,blocks}} o_{n,\cdot}$
38: **end for**
39: **end for**

while Fig. 4 shows a timing diagram of the parallel working units. The input and output channels need to be split into blocks smaller than 32×32, while the image is split into slices of $1024/c_{\text{in}}$ height (lines 1–3). These blocks are indicated as *YodaNN chip block*. Depending on whether the border is zero-padded or not, $\lfloor (h_k - 1)/2 \rfloor$ or $h_k - 1$ columns need to be preloaded (just in case of 1×1 filters no pixels need to be preloaded) (line 6). The same number of pixels are preloaded from one subsequent column, such that a full square of h_k^2 pixels for each input channel is available in the image bank (line 7). After this preloading step, the SoPs start to calculate the partial sums of all 32 output channels while the input channel is changed every cycle (lines 15–20). When the final input channel is reached, the channel summers keep the final sum for all 32 output channels of the current row and column, which are scaled and biased by the scale-bias unit and the final results are streamed out in an interleaved manner (lines 27–33). In case of $n_{\text{out}} = n_{\text{in}}$ (e.g., 32×32) the same number of cycles are needed to stream out the pixels for all output channels as cycles are needed to sum all input channels for the next row, which means that all computational units of the chip are fully utilized. Each row is processed sequentially, then the system switches to the next column, where again the first pixels of the column are preloaded. The filters are circularly right shifted to be aligned to the correct columns. Then, the next column of all output channels are calculated. This procedure is repeated until the whole image and blocks of input and output channels have been processed. Finally, the partial sums for each output channels need to be summed together for every block of input channels (line 37).

We use the same sliding window approach developed in [13] and Fig. 5 shows the implemented sliding window approach. To avoid shifting all images in the image memory to the left for the next column, the right most pixels are inserted at the position of the obsolete pixel, and the weights are shifted instead. To illustrate this, (2) shows the partial convolution for one pixel while the pixels are aligned to the actual column order and (3) shows it when the next column is processed and the weights need to be aligned. To indicate the partial sum, the Frobenius inner product formalism is used, where $\langle \mathbf{A}, \mathbf{B} \rangle_F = \sum_{i,j} a_{ij} b_{ij}$

$$\tilde{o}(2,2) = \left\langle \begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix}, \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} \right\rangle_F \quad (2)$$

$$\tilde{o}(3,2) = \left\langle \begin{bmatrix} x_{14} & x_{12} & x_{13} \\ x_{24} & x_{22} & x_{23} \\ x_{34} & x_{32} & x_{33} \end{bmatrix}, \begin{bmatrix} w_{13} & w_{11} & w_{12} \\ w_{23} & w_{21} & w_{22} \\ w_{33} & w_{31} & w_{32} \end{bmatrix} \right\rangle_F . \quad (3)$$

Equation 3 shows the operands as they are applied to the SoP units. The fourth column which should be the most-right column is in the first column and also the other columns are shifted to the right, thus the weights also needs to be shifted to the right to obtain the correct result. The permutation in algebraic form is formulated

$$\tilde{o}(3,2) = \left\langle \begin{bmatrix} x_{14} & x_{12} & x_{13} \\ x_{24} & x_{22} & x_{23} \\ x_{34} & x_{32} & x_{33} \end{bmatrix}, \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} \cdot P \right\rangle_F \quad (4)$$

where $P = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$ is the permutation matrix.

B. BinaryConnect Approach

In this paper we present a CNN accelerator based on BinaryConnect [22]. With respect to an equivalent 12-bit version, the first major change in architecture are the weights

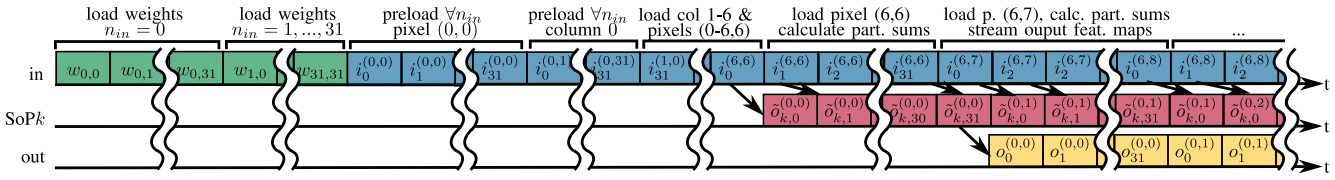


Fig. 4. Timing diagram of the operating scheme: input stream, SoP k's operations, and output stream after accumulation.

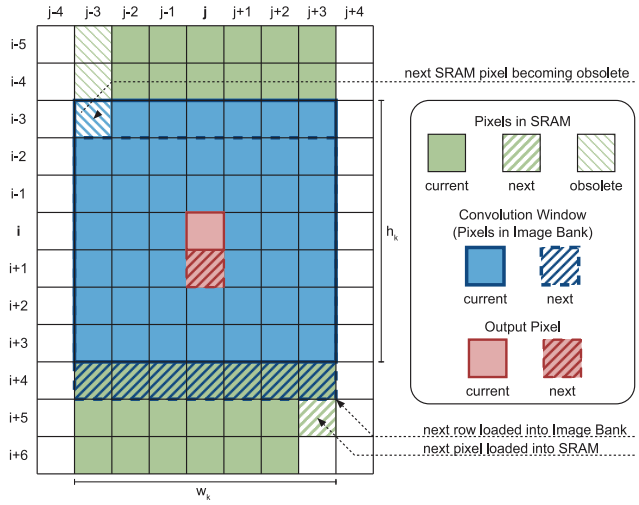


Fig. 5. Sliding window approach of the image memory.

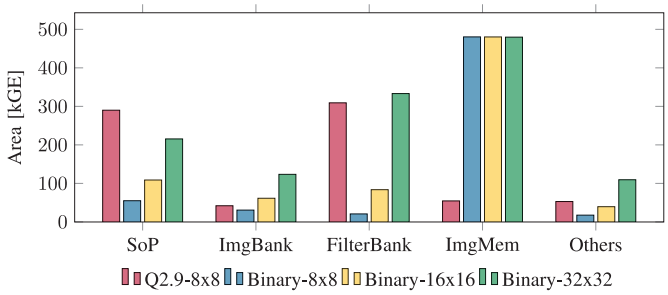


Fig. 6. Area breakdown for fixed-point and several binary architectures.

which are reduced to a binary value $w_{k,n} \in \{-1, 1\}$ and remapped by the following equation:

$$f : \{-1, 1\} \to \{0, 1\},\ y \mapsto \begin{cases} 0 & \text{if } z = -1 \\ 1 & \text{if } z = 1. \end{cases} \tag{5}$$

The size of the filter bank decreases thus from $n_{\text{ch}}^2 \cdot h_k^2 \cdot 12 = 37\,632$ bits to $n_{\text{ch}}^2 \cdot h_k^2 \cdot 1 = 3136$ bits in case of the 12-bit MAC architecture with 8×8 channels and 7×7 filters that we consider as baseline. The 12×12-bit multipliers can be substituted by two's-complement operations and multiplexers, which reduce the "multiplier" and the adder tree size, as the products have a width of 12 bits instead of 24. The SoP is fed by a 12-bit and 7×7 pixel sized image window and 7×7 binary weights. Fig. 6 shows the impact on area while moving from 12-bit MACs to the binary connect architectures. Considering that with the 12-bit MAC implementation 40% of the total chip area is used for the filter bank and another 40% are needed for the 12×12-bit multipliers and the accumulating adder trees, this leads to a significant reduction in area cost and complexity. In fact the area of the conventional SoP unit could be reduced by $5.3\times$ and the filter bank by $14.9\times$ when moving from the Q2.9 to the binary version. The impact on the filter bank is straightforward as 12 times less bits need to be saved compared to the Q2.9, but also the SoP shrinks, as the 12×12-bit multipliers are replaced with 2s complement operation units and multiplexers and the adder tree needs to support a smaller dynamic range, thanks to the smaller products, since the critical path is reduced as well. It is possible to reduce voltage while still keeping the same operating frequency and thus improving the energy efficiency even further.

C. Latch-Based SCM

An effective approach to optimize energy efficiency is to adapt the supply voltage of the architecture according to the performance requirements of the application. However, the potential of this approach is limited by the presence of SRAMs for implementation of image memory, which bounds the voltage scalability to 0.8 V (in 65-nm CMOS technology). To overcome this limitation, we replace the SRAM-based image memory with a latch-based SCMs taking advantage of the area savings achieved through adoption of binary SoPs.

Indeed, although SCMs are more expensive in terms of area (Fig. 6), they are able to operate in the whole operating range of the technology (0.6 – 1.2 V) and they also feature significantly smaller read/write energy [26] at the same voltage. To reduce the area overhead of the SCMs and improve routability we propose a multibanked implementation, where the image memory consists of a latch array organized in 6×8 blocks of 128 rows of 12-bit values, as described in Fig. 7. A predecoding logic, driven by the controller of the convolutional accelerator addresses the proper bank of the array every cycle, generating the local write and read enable signals, the related address fields, and propagating the input pixels to the banks and the current pixels to the SoP unit. During a typical CNN execution, every cycle, six SCMs banks are read, and one is written, according to the image memory access pattern described in Fig. 5.

The SCMs are designed with a hierarchical clock gating and address/data silencing mechanisms as shown in Fig. 8, so that when a bank is not accessed the whole latch array consumes no dynamic power. Every SCM block consists of a 12-bit $\times$ 128 rows array of latches, a data-in write path, and a read-out path. To meet the requirements of the application, the SCM banks are implemented with a two-ported, single-cycle latency architecture with input data and read address sampling. The write path includes data-in sampling registers, and a two-level clock gating scheme for minimizing the dynamic power of the

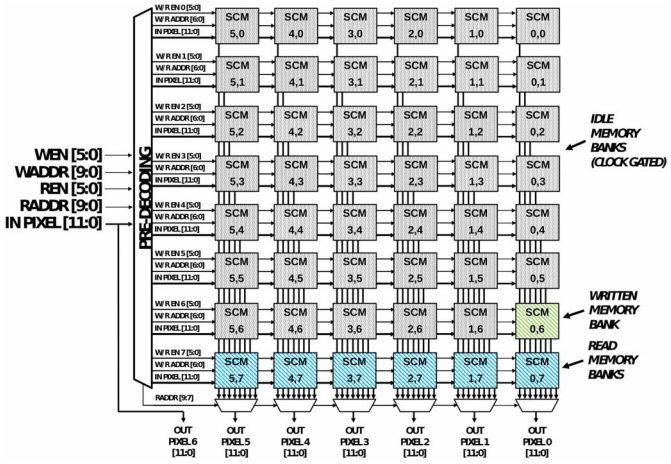


Fig. 7. Image memory architecture.

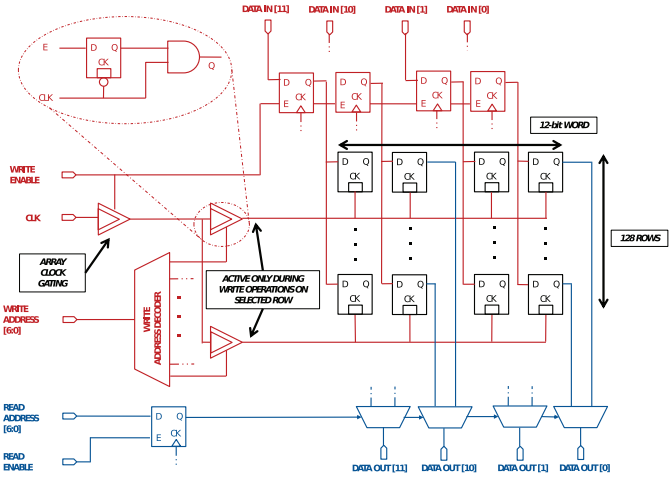


Fig. 8. Block diagram of one SCM bank.

clock path to the storage latches. The array write enable port drives the global clock gating cell, while the row clock gating cells are driven by the write address one-hot decoder. The readout path is implemented with a read address register with clock gating driven by a read enable signal, and a static multiplexer tree, which provides robust and low power operation, and enables dense and low congestion layout.

Thanks to this optimized architecture based on SCMs, only up to 7 out of 48 banks of SCM banks consume dynamic power in every cycle, reducing power consumption of the memory by 3.25× at 1.2 V with respect to a solution based on SRAMs [15], while extending the functional range of the whole convolutional engine down to 0.6 V which is the voltage limit of the standard cells in UMC 65-nm technology chosen for implementation [51].

D. Considering I/O Power in Energy Efficiency

I/O power is a primary concern of convolutional accelerators, consuming even more than 30% of the overall chip power [50]. As we decrease the computational complexity by the binary approach, the I/O power gets even more critical. Fortunately, if the number of output channels is increased,

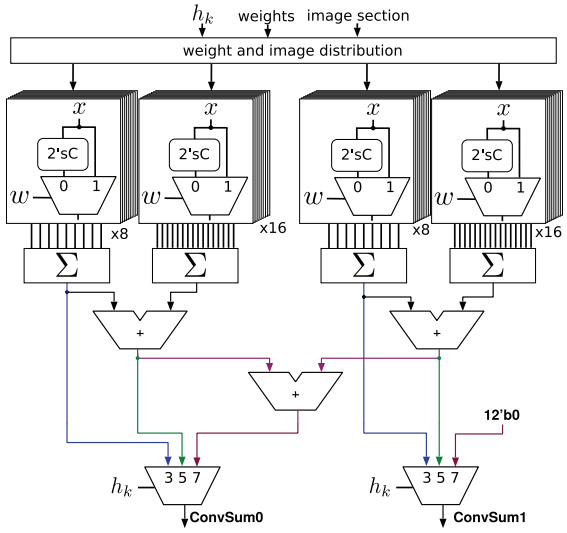


Fig. 9. Adder tree in the SoP unit: different colors are showing the data paths for 3 × 3, 5 × 5, and 7 × 7 kernels are indicated. The operands of the unused adders are silenced, but not indicated in the figure.

more operations can be executed on the same data, which reduces the needed bandwidth and pad power consumption. The other advantage with having more SoP units on-chip is throughput which is formulated in the following equation:

$$\Theta = 2 \cdot \left(n_{\text{filt}}^2 \cdot n_{\text{ch}}\right) \cdot f. \tag{6}$$

With this in mind, we increased the number of input and output channels from 8 × 8 to 16 × 16 and 32 × 32 which provides an ideal speed-up of throughput by 2× and 4×, respectively.

E. Support for Different Filter Sizes, Zero-Padding, Scaling and Biasing

Adapting filter size to the problem provides an effective way to improve the flexibility and energy efficiency of the accelerator when executing CNNs with different requirements. Although the simplest approach is to zero-pad the filters, this is not feasible in the presented binary connect architecture, as the value 0 is mapped to −1. A more energy-efficient approach tries to reuse parts of the architecture. We present an architecture where we reuse the binary multipliers for two 3 × 3, two 5 × 5, or one 7 × 7 filters. In this paper we limit the number of output channels per SoP unit to two as we are limited in output bandwidth. With respect to our baseline architecture, supporting only 7 × 7 filters, the number of binary operators and the weights per filter is increased from 49 to 50, such that two 5 × 5 or one 7 × 7 filter fits into one SoP unit. In case a filter size of 3 × 3 or 5 × 5 is used, the image from the image bank is mapped to the first 25 input image pixels, and the latter 25 and are finally accumulated in the adjusted adder tree, which is drawn in Fig. 9. With this scheme, $n_{\text{ch}} \times 2n_{\text{ch}}$ channels for 3 × 3 and 5 × 5 filters can be calculated, which improves the maximum bandwidth and energy efficiency for these two cases. The unused 2s complement-and-multiplex operands (binary multipliers) and the related part of the adder tree are silenced and clock-gated to reduce switching, therefore keeping the power dissipation as low as possible.

To support also different kernel sizes, we provide the functionality to zero-pad the unused columns from the image memory and the rows from the image bank instead of zeroing the weights which does not make sense with binary weights. This allows us to support kernels of size 1×1, 2×2, 4×4, and 6×6 as well. The zero-padding is also used to add zeros to image borders: e.g., for a 7×7 convolution the first three columns and first three rows of the fourth column is preloaded. The three columns right to the initial pixel and the three rows on top of the pixel are zeroed the same way as described before and thus have not to be loaded onto the chip.

Finally, the system supports channel scaling and biasing which are common operations (e.g., in batch normalization layer) in neural networks which can be calculated efficiently. As described in the previous section up to two output channels are calculated in parallel in every SoP unit, therefore the SoP saves also two scaling and two biasing values for these different output channels. As the feature maps are kept in maximum precision on-chip, the channel summers' output Q7.9 fixed-point values, which are than multiplied with the Q2.9 formatted scaling factor and added to the Q2.9 bias and finally the Q10.18 output is resized with saturation and truncation to the initial Q2.9 format. With the interleaved data streaming, these operations are just needed once per cycle or twice when the number of output channels are doubled (e.g., $k = 3 \times 3$).

IV. Results

A. Computational Complexity and Energy Efficiency Measure

Research in the field of deep learning is done on a large variety of systems, such that platform-independent performance metrics are needed. For computational complexity analysis the total number of multiplications and additions has been used in other publications [13], [16], [42], [52]. For a CNN layer with n_{in} input channels and n_{out} output channels, a filter kernel size of $h_k \times w_k$, and an input size of $h_{\text{im}} \times w_{\text{im}}$, the computational complexity to process one frame can be calculated as follows:

$$\#\text{Op} = 2n_{\text{out}}n_{\text{in}}h_k w_k(h_{\text{in}} - h_k + 1)(w_{\text{in}} - h_k + 1). \quad (7)$$

The factor of 2 considers additions and multiplications as separate arithmetic operations (Op), while the rest of the equation calculates the number of multiply accumulate operations MACs. The two latter factors $(h_{\text{in}} - h_k + 1)$ and $(w_{\text{in}} - h_k + 1)$ are the height and width of the output channels including the reduction at the border in case no zero-padding was applied. Memory accesses are not counted as additional operations. The formula does not take into account the amount of operations executed when applying zero-padding. In the following evaluation, we will use the following metrics.

1) Throughput $\Theta = (\#\text{Op based on (7)})/t$ [GOp/s].
2) *Peak Throughput:* Theoretically reachable throughput. This does not take into account idling, cache misses, etc.
3) Energy efficiency $H_E = \Theta/P$ [TOp/s/W].
4) Area efficiency $H_A = \Theta/A$ [GOp/s/MGE].

Furthermore, we will introduce some efficiency metrics to allow for realistic performance estimates, as CNN layers have varying numbers of input and output channels and image sizes vary from layer to layer

$$\Theta_{\text{real}} = \Theta_{\text{peak}} \cdot \prod_i \eta_i. \quad (8)$$

1) Tiling: The number of rows are limited by the image window memory, which accommodates $h_{\text{max}} \cdot n_{\text{ch,in}}$ words of $w_k \cdot 12$ bit, storing a maximum of h_{max} rows per input channel. In case the full image height does not fit into the memory, it can be split into several image tiles which are then processed consecutively. The penalty are the $(h_k - 1)$ rows by which the tiles need to vertically overlap and thus are loaded twice. The impact on throughput can be determined by the tiling efficiency

$$\eta_{\text{tile}} = \frac{h_{\text{im}}}{h_{\text{im}} + \left(\left\lceil \frac{h_{\text{im}}}{h_{\text{max}}} \right\rceil - 1\right)(h_k - 1)}. \quad (9)$$

2) (Input) Channel Idling: The number of output and input channels usually does not correspond to the number of output and input channels processed in parallel by this core. The output and input channels are partitioned into blocks of $n_{\text{ch}} \times n_{\text{ch}}$. Then the outputs of these blocks have to be summed up pixel-wise outside the accelerator.

In the first few layers, the number of input channels n_{in} can be smaller than the number of output channels n_{out}. In this case, the output bandwidth is limiting the input bandwidth by a factor of η_{chIdle}

$$\eta_{\text{chIdle}} = \frac{n_{\text{in}}}{n_{\text{out}}}. \quad (10)$$

Note that this factor only impacts throughput, not energy efficiency. Using less than the maximum available number of input channels only results in more cycles being spent idling, during which only a negligible amount of energy (mainly leakage) is dissipated.

3) Border Considerations: To calculate one pixel of an output channel, at least h_k^2 pixels of each input channel are needed. This leads to a reduction of $(1/2)(h_k - 1)$ pixels on each side. While in some cases this is acceptable, many and particularly deep CNNs perform zero-padding to keep a constant image size, adding an all-zero halo around the image. In case of zero-padding, $[(h_k - 1)/2]$ columns need to be preloaded, this introduces latency, but does not increase idleness as the same number of columns need to be processed after the last column where in the meantime the first columns of the next image can be preloaded to the image and therefore $\eta_{\text{border}} = 1$. For nonzero padded layers, the efficiency is reduced by the factor

$$\eta_{\text{border,non-zero-padded}} = \frac{h_k - 1}{w_{\text{im}}} \cdot \frac{h_k - 1}{h_{\text{im}}}. \quad (11)$$

B. Experimental Setup

To evaluate the performance and energy metrics of the proposed architecture and to verify the correctness of the generated results, we developed a testbench, which generates the control signals of the chip, reads the filters and the input images from a raw file, and streams the data to the chip. The output is monitored and compared to the expected

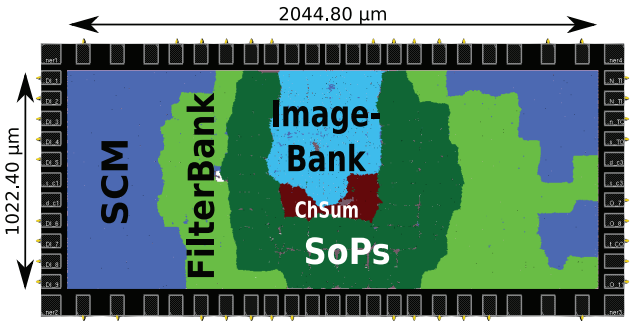


Fig. 10. Floorplan of YodaNN with a 9.2 KiB SCM memory computing 32 output channels in parallel.

output feature maps which are read from a file, too. To calculate the expected responses we have implemented a bit-true quantized spatial convolution layer in Torch which acts as a golden model. The power results are based on post place & route results of the design. The design was synthesized with Synopsys Design Compiler J-2014.09-SP4, while place and route was performed with Cadence Innovus 15.2. The UMC 65-nm standard cell libraries used for implementation were characterized using Cadence Liberate 12.14 in the voltage range 0.6 – 1.2 V, and in the typical process corner at the temperature of 25 °C. The power simulations were performed with Synopsys PrimePower 2012.12, based on value change dump files extracted from simulations of real-life workloads running on the post place and route netlist of the design. These simulations were done with the neural network presented in [50] on the Stanford backgrounds data set [53] (715 images, 320 × 240 RGB, scene-labeling for various outdoor scenes), where every pixel is assigned with one of eight classes, i.e., sky, tree, road, grass, water, building, mountain, and foreground object. The I/O power was approximated by power measurements on chips of the same technology [15] and scaled to the actual operating frequency of YodaNN.

The final floorplan of YodaNN is shown in Fig. 10. The area is split mainly among the SCM memory with 480 kGE, the binary weights filter bank with 333 kGE, the SoP units with 215 kGE, the image bank with 123 kGE and the area distribution is drawn in Fig. 6. The core area is 1.3 MGE (1.9 mm^2). The chip runs at a maximum frequency of 480 MHz@1.2 V and 27.5 MHz@0.6 V.

C. Fixed-Point Versus YodaNN

In this section, we compare a fixed-point baseline implementation with a binary version with fixed filter kernel size of 7 × 7 and 8 × 8 channels including an SRAM for input image storage. The results are summarized in Table I. The reduced arithmetic complexity and the replacement of the SRAM by a latch-based memory shortened the critical path delay. Three pipeline stages between the memory and the channel summers were used in the fixed-point baseline version could be reduced to one pipeline stage. The peak throughput could still be increased from 348 GOp/s to 377 GOp/s at a core voltage of 1.2 V and the core power was reduced by 79 % to 39 mW, which leads to a 5.1× better core energy efficiency and 1.3× better core area efficiency. As UMC 65-nm technology SRAMs fail below 0.8 V, we can get even better results by reducing the supply voltage to 0.6 V thanks to our SCM implementation. Although the peak throughput drops to 15 GOp/s, the core power consumption is reduced to 260 µW, and core energy efficiency rises to 59 TOp/s/W, which is an improvement of 11.6× compared to the fixed-point architecture at 0.8 V.

Fig. 11 shows the throughput and energy efficiency of YodaNN with respect to the baseline architecture for different voltage supplies, while Fig. 12 shows the breakdown of the core power an the operating frequency of 400 MHz. Comparing the two 8×8 channels variants (fixed-point and binary weights), the power consumption was reduced from 185 to 39 mW, where the power could be reduced by 3.5× in the SCM, 4.8× in the SoP units and 31× in the filter bank. Although the power consumption of the core increases

TABLE I
FIXED-POINT Q2.9 VERSUS BINARY ARCHITECTURE 8 × 8

Architecture	Q2.9[a]	Bin.	Q2.9[a]	Bin.	Bin.
Supply (V)	1.2	1.2	0.8	0.8	0.6
Peak Throughput (GOp/s)	348	377	131	149	15
Avg. Power Core (mW)	185	39	31	5.1	0.26
Avg. Power Device (mW)	580	434	143	162	15.54
Core Area (MGE)	0.72	0.60	0.72	0.60	0.60
Efficiency metrics					
Energy Core (TOp/s/W)	1.88	9.61	4.26	29.05	58.56
Energy Device (TOp/s/W)	0.60	0.87	0.89	0.92	0.98
Area Core (GOp/s/MGE)	487	631	183	247	25
Area Dev. (GOp/s/MGE)	161	175	61	69	7.0

[a] A fixed-point version with SRAM is used as baseline comparison and 8×8 channels and 7×7 filters.

TABLE II
DEVICE ENERGY EFFICIENCY FOR DIFFERENT FILTERS AND ARCHITECTURES

Archit.	Q2.9	8×8	16×16	32×32	32^2 (fixed)	
7×7	600	856	1611	2756	3001	[GOp/s/W]
5×5		611	1170	2107		[GOp/s/W]
3×3		230	452	859		[GOp/s/W]

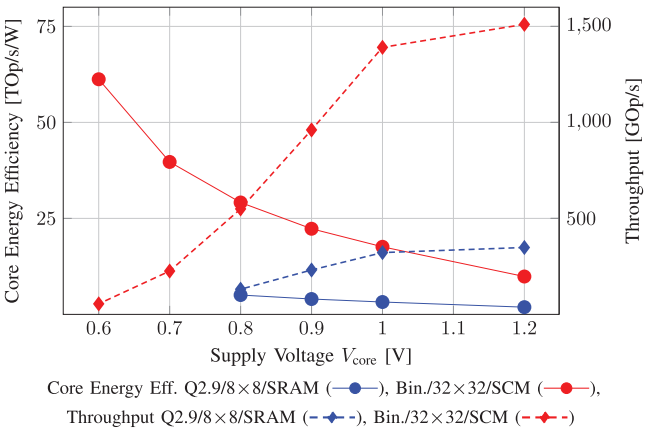


Fig. 11. Comparison of core energy efficiency and throughput for the baseline architecture (fixed-point Q2.9, SRAM, 8×8 channels, fixed 7×7 filters) with final YodaNN (binary, SCM, 32×32 channels, supporting several filters).

TABLE III
EVALUATION ON SEVERAL WIDELY KNOWN CNNS IN THE HIGH-EFFICIENCY CORNER

Network	L	h_k px	w px	h px	n_{in}	n_{out}	$\times$	η_{tile}	η_{Idle}	$\tilde{P}_{real}$	Θ_{real} GOp/s	$EnEff$ TOp/s/W	#MOp	t ms	E µJ
BinaryConnect Cifar-10 [22]	1	3	32	32	3	128	1	1.00	0.09	0.35	1.9	16.0	7	3.8	0.4
	2	3	32	32	128	128	1	1.00	1.00	1.00	20.1	59.2	302	15.0	5.1
	3	3	16	16	128	256	1	1.00	1.00	1.00	20.1	59.2	151	7.5	2.6
	4	3	16	16	256	256	1	1.00	1.00	1.00	20.1	59.2	302	15.0	5.1
	5	3	8	8	256	512	1	1.00	1.00	1.00	20.1	59.2	151	7.5	2.6
	6	3	8	8	512	512	1	1.00	1.00	1.00	20.1	59.2	302	15	5.1
	7	FC	4	4	512	1024	1						16		
	8	FC	1	1	1024	1024	1						2		
	9	SVM	1	1	1024	10	1						0.0		
BinaryConnect SVHN [22]	1	3	32	32	3	128	1	1.00	0.09	0.35	1.9	16.0	7	3.8	0.4
	2	3	16	16	128	256	1	1.00	1.00	1.00	20.1	59.2	151	7.5	2.6
	3	3	8	8	256	512	1	1.00	1.00	1.00	20.1	59.2	151	7.5	2.6
	4	FC	4	4	512	1024	1						16		
AlexNet ImageNet[2]	1ab[2]	6	224	224	3	48	4	0.95	0.09	0.35	1.4	12.1	520	364.7	42.9
	1cd[2]	5	224	224	3	48	4	0.9	0.07	0.35	3.55	11.8	361	101.7	30.5
	2	5	55	55	48	128	2	0.93	0.75	1.00	39.1	45.2	929	23.8	20.6
	3	3	27	27	128	192	2	1.00	1.00	1.00	20.1	59.2	322	16.0	5.4
	4	3	13	13	192	192	2	1.00	1.00	1.00	20.1	59.2	112	5.6	1.9
	5	3	13	13	192	128	2	1.00	1.00	1.00	20.1	59.2	75	3.7	1.3
	7	FC	13	13	256	4096	1						354		
	8	FC	1	1	4096	4096	1						34		
	9	FC	1	1	4096	1000	1						8		
ResNet-18/34 ImageNet[4]	1	7	224	224	3	64	1	0.86	0.09	0.35	4.4	15.1	236	53.3	15.7
	2-5	3	56	56	64	64	4/6	0.95	1.00	1.00	19.1	56.2	231	11.9	4.0
	6	3	28	28	64	128	1	0.97	1.00	1.00	19.4	57.2	116	5.7	2.0
	7-9	3	28	28	128	128	3/7	0.97	1.00	1.00	19.4	57.2	231	11.5	3.9
	10	3	14	14	128	256	1	1.00	1.00	1.00	20.1	59.2	116	5.7	2.0
	11-13	3	14	14	256	256	3/11	1.00	1.00	1.00	20.1	59.2	231	11.5	3.9
	14	3	7	7	256	512	1	1.00	1.00	1.00	20.1	59.2	116	5.7	2.0
	15-17	3	7	7	512	512	3	1.00	1.00	1.00	20.1	59.2	231	11.5	3.9
	18	FC	7	7	512	1000	1						200		
VGG-13/19 ImageNet[32]	1	3	224	224	3	64	1	0.95	0.09	0.35	1.9	15.2	173	91.9	11.4
	2	3	224	224	64	64	1	0.95	1.00	1.00	19.1	56.2	3699	193.6	65.8
	3	3	112	112	64	128	1	0.95	1.00	1.00	19.1	56.2	1850	96.8	32.9
	4	3	112	112	128	128	1	0.95	1.00	1.00	19.1	56.2	3699	193.6	65.8
	5	3	56	56	128	256	1	0.97	1.00	1.00	19.4	57.2	1850	95.2	32.4
	6	3	56	56	256	256	1/3	0.97	1.00	1.00	19.4	57.2	3699	190.3	64.7
	7	3	28	28	256	512	1	1.00	1.00	1.00	20.1	59.2	1850	91.9	31.2
	8	3	28	28	512	512	1/3	1.00	1.00	1.00	20.1	59.2	3699	183.8	62.5
	9-10	3	14	14	512	512	2/4	1.00	1.00	1.00	20.1	59.2	925	45.9	15.6
	11	FC	14	14	256	4096	1						411		
	12	FC	1	1	4096	4096	1						34		
	13	FC	1	1	4096	1000	1						8		

Legend: L: layer, h_k: kernel size, w: image width, h: image height, n_i: input channels, n_o: output channels, $\times$: quantity of this kind of layer, η_{tile}: tiling efficiency, η_{chIdle}: channel idling efficiency, $\tilde{P}_{real}$ Normalized Power consumption in respect to active convolving mode, Θ_{real}: actual throughput, EnEff: Actual Energy Efficiency, #MOp: Number of operations (additions or multiplications, in millions), t: time, E: needed processing energy

by 3.32× when moving from 8 × 8 to 32 × 32 channels, the throughput increases by 4×, improving energy efficiency by 20%. Moreover, taking advantage of more parallelism, voltage and frequency scaling can be exploited to improve energy efficiency for a target throughput. The support for different kernel sizes significantly improves the flexibility of the YodaNN architecture, but increases the core area by 11.2%, and the core power by 38% with respect to a binary design supporting 7×7 kernels only. The scale-bias unit occupies another 2.5 kGE area and consumes 0.4 mW at a supply voltage of 1.2 V and a operating frequency of 480 MHz. When I/O power is considered, increasing the number of channels is more beneficial, since we can increase the throughput while the total device power does not increase at the same rate. We estimate a fixed contribution of 328 mW for the I/O power at 400 MHz. Table II provides an overview of the device energy efficiency for different filter kernel sizes at 1.2 V core and 1.8 V pad supply. The device energy efficiency raises from 856 GOps/s/W in the 8 × 8 architecture to 1611 in the 16 × 16 and to 2756 in the 32 × 32.

D. Real Applications

For a comparison based on real-life CNNs, we have selected several state-of-the-art networks which exploit binary weights. This includes the CNNs from the BinaryConnect paper for

[2]The 11 × 11 kernels are split into two 6 × 6 and two 5 × 5 kernels as described in Section IV-D.

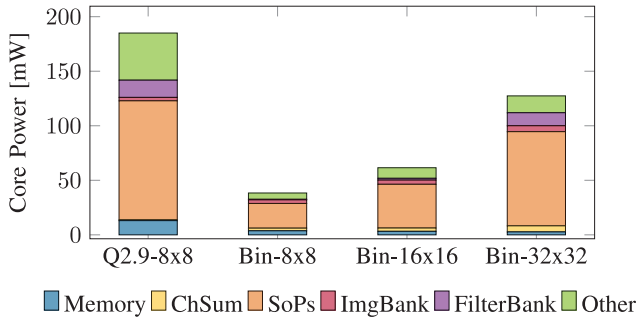


Fig. 12. Core power breakdown for fixed-point and several binary architectures.

Cifar-10 and SVHN [22], and the well-known networks VGG-13, VGG-19 [32], ResNet-18, ResNet-34 [4], and AlexNet [2], which were successfully implemented with binary weights by Rastegari *et al.* [23] (not XNOR-net). The layer configurations and the related metrics are summarized in Table III. As described in Section III-A, the layers are split into blocks of $n_{\text{in}} \times n_{\text{out}} = 32 \times 32$ channels in case of a kernel size of $h_k^2 = 7^2$ and $n_{\text{in}} \times n_{\text{out}} = 32 \times 64$ elsewhere. The first layers have a high idle rate, but the silenced SoP units consume roughly no power. To account for this we introduce $\tilde{P}_{\text{real}} = P_{\text{eff}}/P_{\text{max}}$ which is calculated. The first layer of AlexNet uses 11×11 filters and needs to be split into smaller kernels. We split them into two filters of 6×6 (top-left, bottom-right) and two filters of 5×5 (bottom-left, top-right), where the center pixel is overlapped by both 6×6 kernels. By choosing the value for the overlapping weight appropriately, it is possible to prevent the need of additional 1×1 convolutions: if the original weight is 1, the overlapping weight of both 6×6 kernels are chosen to be 1, otherwise -1 is assigned to one of them and 1 to the other. Instead of 1×1 convolutions, just the sum of the identities of all input channels needs to be subtracted. The summing of the contributions and subtracting of the identities is done off-chip.

Table IV gives an overview of the energy efficiency, throughput, actual frame rate and total energy consumption for calculating the convolutions, including channel biasing and scaling in the energy-optimal configuration (at 0.6 V). Table V shows the same metrics and CNNs for the high-throughput setting at 1.2 V. It can be noticed that in the energy-optimal operating point, the achieved throughput is about half of the maximum possible throughput of 55 GOp/s for most of the listed CNNs. This can be attributed to the smaller-than-optimal filter size of 3×3, which is frequently used and limits the throughput to about 20 GOp/s. However, note that the impact on peak energy-efficiency is only minimal with 59.20 instead of 61.23 GOp/s/W.

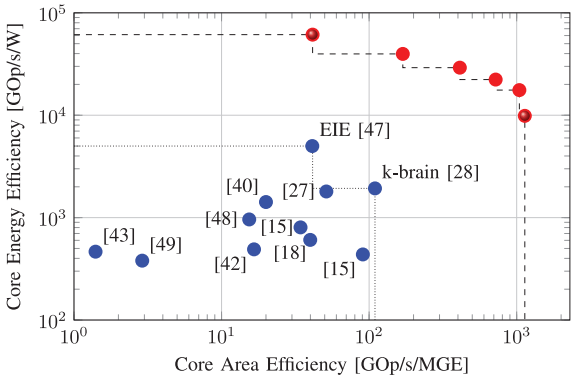
The average energy efficiency of the different networks is within the range from 48.1 to 56.7 TOp/s/W, except for AlexNet which reaches 14.1 TOp/s/W due to the dominant first layer which requires a high computational effort while leaving the accelerator idling for a large share of the cycles because of the small number of input channels. The fourth column in Tables IV and V show the frame rate which can be processed by YodaNN excluding the fully connected layers and the chip

TABLE IV
OVERVIEW OF SEVERAL NETWORKS IN AN ENERGY OPTIMAL USE CASE ($V_{\text{CORE}} = 0.6$ V) ON A YODANN ACCELERATOR

Network	img size $h_{in} \times w_{in}$	Avg. EnEff TOp/s/W	$\bar{\Theta}$ GOp/s	Θ FPS	Energy µJ
BC-Cifar-10	32×32	56.7	19.1	15.8	21
BC-SVHN	32×32	50.6	16.5	53.2	6
AlexNet	224×224	14.1	3.3	0.5	352
ResNet-18	224×224	48.1	16.2	4.5	73
ResNet-34	224×224	52.5	17.8	2.5	136
VGG-13	224×224	54.3	18.2	0.8	398
VGG-19	224×224	55.9	18.9	0.5	684

TABLE V
OVERVIEW OF SEVERAL NETWORKS IN A THROUGHPUT OPTIMAL USE CASE ($V_{\text{CORE}} = 1.2$ V) ON A YODANN ACCELERATOR

Network	img size $h_{in} \times w_{in}$	Avg. EnEff TOp/s/W	$\bar{\Theta}$ GOp/s	Θ FPS	Energy µJ
BC-Cifar-10	32×32	8.6	525	435	137
BC-SVHN	32×32	7.7	454	1429	36
AlexNet	224×224	2.2	90	14	2244
ResNet-18	224×224	7.3	446	125	478
ResNet-34	224×224	8.0	495	68	889
VGG-13	224×224	8.3	502	22	2609
VGG-19	224×224	8.5	520	13	4482



SoA ASICs (●), YodaNN (1.2–0.6V) (●), YodaNN (1.2 & 0.6V corner) (●), previous Pareto front (·····), our Pareto front (– – –)

Fig. 13. Core area efficiency versus core energy efficiency for state-of-the-art CNN accelerators.

configuration. In the throughput optimal case, the achieved frame rate is between 13.3 (for VGG-19) and 1428 FPS (for the BinaryConnect-SVHN network) with a chip power of just 153 mW. In the maximum energy efficiency corner YodaNN achieves a frame rate between 0.5 and 53.2 FPS at a power of 895μW.

E. Comparison With State-of-the-Art

In Section II, the literature from several software and architectural approaches have been described. The 32×32 channel YodaNN is able to reach a peak throughput of 1.5 TOp/s which outperforms NINEX [27] by a factor of 2.7. In core energy efficiency the design outperforms k-Brain, NINEX by 5× and more. If the supply voltage is reduced to 0.6 V, the throughput decreases to 55 GOp/s but the energy efficiency rises to 61.2 TOp/s, which is more than an

order-of-magnitude improvement over the previously reported results [27], [28], [40]. The presented architecture also outperforms the compressed neural network accelerator EIE in terms of energy efficiency by 12× and in terms of area efficiency by 28×, even though they assume a very high degree of sparsity with 97% zeros [47]. Fig. 13 gives a quantitative comparison of the state-of-the-art in energy efficiency and area efficiency. For the sweep of voltages between 1.2 and 0.6 V, YodaNN builds a clear Pareto front over the state of the art.

V. Conclusion

We have presented a flexible, energy-efficient and performance scalable CNN accelerator. The proposed architecture is the first ASIC design exploiting recent results on binary-weight CNNs, which greatly simplifies the complexity of the design by replacing fixed-point MAC units with simpler complement operations and multiplexers without negative impact on classification accuracy. To further improve energy efficiency and extend the performance scalability of the accelerator, we have implemented latch-based SCMs for on-chip data storage to be able to scale down the operating voltage even further. To add flexibility, we support seven different kernel sizes: 1×1, 2×2, $\ldots$, 7×7. This enables efficient evaluation of a large variety of CNNs. Even though this added flexibility introduces a 29% reduction in energy efficiency, an outstanding overall energy efficiency of 61 TOp/s/W is achieved. The proposed accelerator surpasses state-of-the-art CNN accelerators by 2.7× in peak performance with 1.5 TOp/s, by 10× in peak area efficiency with 1.1 TOp/s/MGE and by 32× peak energy efficiency with 61.2 TOp/s/W. YodaNN's power consumption at 0.6 V is 895 μW with an average frame rate of 11 FPS for state-of-the-art CNNs and 16.8 FPS for ResNet-34 at 1.2 V.

References

[1] G. Lucas. (2016). *Yoda*. [Online]. Available: www.starwars.com/databank/yoda

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1106–1114.

[3] R. Wu, S. Yan, Y. Shan, Q. Dang, and G. Sun, "Deep image: Scaling up image recognition," arXiv preprint arXiv:1501.02876, Jan. 2015.

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," arXiv preprint arXiv:1512.03385, Dec. 2015.

[5] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "DeepFace: Closing the gap to human-level performance in face verification," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Columbus, OH, USA, Jun. 2014, pp. 1701–1708.

[6] A. Y. Hannun *et al.*, "Deep speech: Scaling up end-to-end speech recognition," arXiv preprint arXiv:1412.5567, Dec. 2014.

[7] J. Weston, S. Chopra, and A. Bordes, "Memory networks," arXiv:1410.3916, Oct. 2014.

[8] J. Weston, "Dialog-based language learning," arXiv preprint arXiv:1604.06045, Apr. 2016.

[9] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb 2015.

[10] M. Zastrow, "Machine outsmarts man in battle of the decade," *New Sci.*, vol. 229, no. 3065, p. 21, 2016.

[11] A. Coates *et al.*, "Deep learning with cots HPC systems," in *Proc. 30th Int. Conf. Mach. Learn. (ICML)*, vol. 28. Atlanta, GA, USA, May 2013, pp. 1337–1345.

[12] C. Farabet, C. Couprie, L. Najman, and Y. LeCun, "Learning hierarchical features for scene labeling," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 8, pp. 1915–1929, Aug. 2013.

[13] L. Cavigelli, M. Magno, and L. Benini, "Accelerating real-time embedded scene labeling with convolutional networks," in *Proc. 52nd Annu. Design Autom. Conf. (DAC)*, San Francisco, CA, USA, 2015, pp. 1–6.

[14] K. Ovtcharov *et al.*, "Accelerating deep convolutional neural networks using specialized hardware," *Microsoft Res. Whitepaper*, vol. 2, no. 11, 2015.

[15] L. Cavigelli and L. Benini, "Origami: A 803 GOp/s/W convolutional network accelerator," arXiv preprint arXiv:1512.04295, Jan. 2016.

[16] C. Farabet *et al.*, "NeuFlow: A runtime reconfigurable dataflow processor for vision," in *Proc. CVPR WORKSHOPS*, Colorado Springs, CO, USA, Jun. 2011, pp. 109–116.

[17] F. Conti and L. Benini, "A ultra-low-energy convolution engine for fast brain-inspired vision in multicore clusters," in *Proc. Design Autom. Test Europe Conf. Exhibit.*, Grenoble, France, 2015, pp. 683–688.

[18] Z. Du *et al.*, "ShiDianNao: Shifting vision processing closer to the sensor," in *Proc. ACM SIGARCH Comput. Archit. News*, Portland, OR, USA, 2015, pp. 92–104.

[19] W. Qadeer *et al.*, "Convolution engine: Balancing efficiency & flexibility in specialized computing," in *Proc. ISCA*, Tel Aviv, Israel, 2013, pp. 24–35.

[20] W. Sung, S. Shin, and K. Hwang, "Resiliency of deep neural networks under quantization," arXiv preprint arXiv:1511.06488, Nov. 2015.

[21] P. Gysel, M. Motamedi, and S. Ghiasi, "Hardware-oriented approximation of convolutional neural networks," arXiv preprint arXiv:1604.03168, May 2016.

[22] M. Courbariaux, Y. Bengio, and J.-P. David, "BinaryConnect: Training deep neural networks with binary weights during propagations," in *Proc. Adv. Neural Inf. Process. Syst.*, Montreal, QC, Canada, 2015, pp. 3105–3113.

[23] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet classification using binary convolutional neural networks," arXiv preprint arXiv:1603.05279, Mar. 2016.

[24] M. Courbariaux and Y. Bengio, "BinaryNet: Training deep neural networks with weights and activations constrained to +1 or -1," arXiv preprint arXiv:1602.02830, Mar. 2016.

[25] Z. Lin, M. Courbariaux, R. Memisevic, and Y. Bengio, "Neural networks with few multiplications," arXiv preprint arXiv:1510.03009, Feb. 2016.

[26] A. Teman, D. Rossi, P. Meinerzhagen, L. Benini, and A. Burg, "Power, area, and performance optimization of standard cell memory arrays through controlled placement," *ACM Trans. Design Autom. Electron. Syst.*, vol. 21, no. 4, 2016, Art. no. 59.

[27] S. Park *et al.*, "14.1 A 126.1mw real-time natural UI/UX processor with embedded deep-learning core for low-power smart glasses," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, San Francisco, CA, USA, Jan. 2016, pp. 254–255.

[28] S. Park *et al.*, "4.6 A1.93TOPS/W scalable deep learning/inference processor with tetra-parallel MIMD architecture for big-data applications," in *IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers (ISSCC)*, San Francisco, CA, USA, Feb. 2015, pp. 1–3.

[29] L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus, "Regularization of neural networks using dropconnect," in *Proc. 30th Int. Conf. Mach. Learn. (ICML)*, vol. 28. Atlanta, GA, USA, May 2013, pp. 1058–1066.

[30] C.-Y. Lee *et al.*, "Generalizing pooling functions in convolutional neural networks: Mixed, gated, and tree," arXiv preprint arXiv:1509.08985, Sep. 2015.

[31] B. Graham, "Fractional max-pooling," arXiv preprint arXiv:1412.607, Dec. 2014.

[32] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," arXiv preprint arXiv:1409.1556, Sep. 2014.

[33] D. D. Lin, S. S. Talathi, and V. S. Annapureddy, "Fixed point quantization of deep convolutional networks," arXiv preprint arXiv:1511.06393, Nov. 2015.

[34] B. Moons, B. De Brabandere, L. Van Gool, and M. Verhelst, "Energy-efficient ConvNets through approximate computing," arXiv preprint arXiv:1603.06777, Mar. 2016.

[35] X. Wu, "High performance binarized neural networks trained on the ImageNet classification task," arXiv preprint arXiv:1604.03058, Apr. 2016.

[36] P. Merolla, R. Appuswamy, J. Arthur, S. K. Esser, and D. Modha, "Deep neural networks are robust to weight binarization and other non-linear distortions," arXiv preprint arXiv:1606.01981, Jun. 2016.

[37] S. Chintala. (2016). *Convnet-Benchmarks*. [Online]. Available: https://github.com/soumith/convnet-benchmarks

[38] N. Jouppi. (2016). *Google Supercharges Machine Learning Tasks With TPU Custom Chip*. [Online]. Available: https://cloudplatform.googleblog.com/2016/05/Google-supercharges-machine-learning-tasks-with-custom-chip.html
[39] *Ins-03510-c1 Datasheet, Datasheet of Myriad 2 Vision Processor*, Movidius, San Mateo, CA, USA, 2014. [Online]. Available: http://uploads.movidius.com/1441734401-Myriad-2-product-brief.pdf
[40] S. Jaehyeong *et al.*, "14.6 A 1.42TOPS/W deep convolutional neural network recognition processor for intelligent IOE systems," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, San Francisco, CA, USA, Apr. 2016, pp. 264–265.
[41] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, San Francisco, CA, USA, Jan. 2016, pp. 262–263.
[42] P.-H. Pham *et al.*, "NeuFlow: Dataflow vision processing system-on-a-chip," in *Proc. IEEE 55th Int. Midwest Symp. Circuits Syst. (MWSCAS)*, Boise, ID, USA, Aug. 2012, pp. 1044–1047.
[43] B. Reagen *et al.*, "Minerva: Enabling low-power, highly-accurate deep neural network accelerators," in *Proc. 43rd Int. Symp. Comput. Archit. (ISCA)*, Seoul, South Korea, 2016, pp. 267–278.
[44] J. Albericio *et al.*, "Cnvlutin: Ineffectual-neuron-free deep neural network computing," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, Seoul, South Korea, Jun. 2016, pp. 1–13.
[45] V. Gokhale, J. Jin, A. Dundar, B. Martini, and E. Culurciello, "A 240 G-ops/s mobile coprocessor for deep neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops*, Columbus, OH, USA, 2014, pp. 696–701.
[46] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," arXiv preprint arXiv:1510.00149, Oct. 2015.
[47] S. Han *et al.*, "EIE: Efficient inference engine on compressed deep neural network," arXiv preprint arXiv:1602.01528, Feb. 2016.
[48] R. LiKamWa, Y. Hou, J. Gao, M. Polansky, and L. Zhong, "Redeye: Analog convnet image sensor architecture for continuous mobile vision," in *Proc. ISCA*, vol. 43. Seoul, South Korea, 2016, pp. 255–266.
[49] A. Shafiee *et al.*, "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in *Proc. ISCA*, Seoul, South Korea, 2016, pp. 14–26.
[50] L. Cavigelli *et al.*, "Origami: A convolutional network accelerator," in *Proc. 25th Edition Great Lakes Symp. VLSI*, Pittsburgh, PA, USA, 2015, pp. 199–204.
[51] A. Pullini *et al.*, "A heterogeneous multi-core system-on-chip for energy efficient brain inspired vision," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Montreal, QC, Canada, 2016, Art. no. 2910.
[52] T. Chen *et al.*, "DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," *SIGARCH Comput. Archit. News*, vol. 42, no. 1, pp. 269–284, Mar. 2014.
[53] S. Gould, R. Fulton, and D. Koller, "Decomposing a scene into geometric and semantically consistent regions," in *Proc. ICCV*, Kyoto, Japan, 2009, pp. 1–8.

Renzo Andri received the M.Sc. degree in electrical engineering and information technology from ETH Zurich, Zürich, Switzerland, in 2015, where he is currently pursuing the Ph.D. degree with the Integrated System Laboratory.

His current research interests involve the design of low-power hardware accelerators for machine learning applications including CNNs, and studying new algorithmic methods to further increase the energy-efficiency and therefore the usability of ML on energy-restricted devices.

Lukas Cavigelli received the M.Sc. degree in electrical engineering and information technology from ETH Zurich, Zürich, Switzerland, in 2014, where he is currently pursuing the Ph.D. degree with the Integrated Systems Laboratory.

His current research interests include deep learning, computer vision, digital signal processing, and low-power integrated circuit design.

Mr. Cavigelli was a recipient of the Best Paper Award at the 2013 IEEE VLSI-SoC Conference.

Davide Rossi received the Ph.D. degree from the University of Bologna, Bologna, Italy, in 2012.

He has been a Post-Doctoral Researcher with the Department of Electrical, Electronic and Information Engineering, University of Bologna, since 2015, where he is currently an Assistant Professor. His current research interests include energy-efficient digital architectures in the domain of heterogeneous and reconfigurable multicore and many-core systems on a chip, architectures, design implementation strategies, and run-time support to address performance, energy efficiency, and reliability issues of both high end embedded platforms and ultralow-power computing platforms targeting the Internet of Thongs domain. He has published over 30 paper in international peer-reviewed conferences and journals in the above areas.

Luca Benini (F'07) received the Ph.D. degree in electrical engineering from Stanford University, Stanford, CA, USA, in 1997.

He is the Chair of Digital Circuits and Systems with ETH Zurich, Zürich, Switzerland, and a Full Professor with the University of Bologna, Bologna, Italy. He has served as a Chief Architect for the Platform2012 with STMicroelectronics, Grenoble, France. He has published over 700 papers in peer-reviewed international journals and conferences, four books, and several book chapters. His current research interests include energy-efficient system, multicore SoC design, energy-efficient smart sensors, and sensor networks.

Dr. Benini is a fellow of ACM, and a member of the Academia Europaea.