2017 IEEE 23rd International Conference on Parallel and Distributed Systems

COSY: An Energy-Efficient Hardware Architecture for Deep Convolutional Neural Networks based on Systolic Array

Chen Xin, Qiang Chen, Miren Tian, Mohan Ji, Chenglong Zou, Xin'an Wang*, Bo Wang
The Key Laboratory of Integrated Microsystems
Peking University Shenzhen Graduate School, Shenzhen, China
**Email: wangxa@pkusz.edu.cn*

***Abstract*—Deep convolutional neural networks (CNNs) show extraordinary abilities in artificial intelligence applications, but their large scale of computation usually limits their uses on resource-constrained devices. For CNN's acceleration, exploiting the data reuse of CNNs is an effective way to reduce bandwidth and energy consumption. *Row-stationary* (RS) dataflow of Eyeriss is one of the most energy-efficient state-of-the-art hardware architectures, but has redundant storage usage and data access, so the data reuse has not been fully exploited. It also requires complex control and is intrinsically unable to skip over zero-valued inputs in timing.**

In this paper, we present COSY (CNN on Systolic Array), an energy-efficient hardware architecture based on the systolic array for CNNs. COSY adopts the method of systolic array to achieve the storage sharing between processing elements (PEs) in RS dataflow at the RF level, which reduces low-level energy consumption and on-chip storage. Multiple COSY arrays sharing the same storage can execute multiple 2-D convolutions in parallel, further increasing the data reuse in the low-level storage and improving throughput. To compare the energy consumption of COSY and Eyeriss running actual CNN models, we build a process-based energy consumption evaluation system according to the hardware storage hierarchy. The result shows that COSY can achieve an over 15% reduction in energy consumption under the same constraints, improving the theoretical Energy-Delay Product (*EDP*) and Energy-Delay Squared Product (ED^2P) by 1.33× on average. In addition, we prove that COSY has the intrinsic ability for zero-skipping, which can further increase the improvements to 2.25× and 3.83× respectively.

***Keywords*-deep convolutional neural network (CNN), systolic array, hardware architecture, energy-efficiency, zero-skipping.**

I. Introduction

Deep convolutional neural network (CNN) is one of the most useful technologies of deep learning, which nowadays plays an important role in solving advanced abstract cognitive problems, especially the image perception tasks [1–6]. For abstract and complex learning problems, the larger scale of CNN is required to achieve high accuracy, leading to a huge increase in computational complexity, storage capacity and energy consumption. Limited by hardware resources, it is a big challenge to implement the CNN models on some embedded devices such as smartphones and IoT (Internet of Things) devices.

In a normal CNN, over 90% of the operations and running time is for convolutions [7, 8], so optimization for convolution is the key to achieve energy-efficient implementation. A great quantity of data reuse is a notable feature of convolution. On hardware, data reuse at high storage level will result in large bandwidth usage, increasing energy consumption. Accordingly, relocating the data reuse to the lower storage level as much as possible delicately is one of the most common and effective ways to reduce energy consumption [9–12].

Many hardware architectures for CNN on FPGA [8–10, 13–15] and ASIC [16–22] are proposed. Eyeriss [20] is one of the most energy-efficient state-of-the-art hardware architecture. Using *Row-stationary* (RS) dataflow, it exploits local data reuse of filter weights and feature map pixels, reduces data movement of partial sum accumulations [23]. Through careful observation on Eyeriss, we find that its data reuse has not been further exploited, mainly reflects in extra storage access in the *register-file* (RF) storage level. This is because of the lack of storage sharing between PEs, which is difficult to achieve. In Eyeriss, a complex controlling flow is also introduced mainly for data configuration before computing.

On average 44% of the convolutional layer input neuron values are zeros that are ineffectual and can be skipped to reduce computing time and energy consumption [19, 24]. For instance, Cnvlutin [19] is a typical hardware implementation aimed at zero-skipping, improving performance by 1.52× compared with its prototype, DaDianNao [21]. We find that the Eyeriss architecture is intrinsically unable to skip over zero-valued inputs in timing, although it can reduce energy consumption by clock gating.

Our work takes Eyeriss as a reference. In this paper, a CNN hardware architecture based on the systolic array is proposed to further exploit the data reuse at low storage level and achieve zero-skipping intrinsically. The architecture, named as COSY (CNN on Systolic array), retains the energy-efficient and configurable feature of Eyeriss RS dataflow, and uses systolic array to make it possible to share internal storage of PEs, which reduces low-level energy consumption. Multiple COSY arrays sharing the same storage perform multiple 2-D convolution calculations in parallel,

978-1-5386-2129-5/17/31.00 ©2017 IEEE
DOI 10.1109/ICPADS.2017.00034

further increasing the data reuse in the low-level storage and improving throughput.

To evaluate the energy consumption of COSY and Eyeriss running actual CNN models in different array sizes objectively, we build a process-based energy consumption evaluation system according to hardware storage hierarchy, which is similar to the analysis framework proposed by Eyeriss group [23]. The result shows that COSY achieves an over 15% reduction in energy consumption under the same hardware and algorithm constraints, improving theoretical The Energy-Delay Product (*EDP*) [25] and Energy-Delay Squared Product (ED^2P) [26] by 1.33× compared with Eyeriss. In addition, we prove that the COSY architecture has intrinsic support for zero-skipping. This ability can further increase the improvements to 2.25× and 3.83× respectively by skipping zero feature map values in timing.

This paper is organized as follows: Section II introduces the data reuse in CNN hardware implementation and motivates COSY's basic thought of shared internal storage by analyzing the redundant data access and storage of Eyeriss at the RF level. Section III presents the COSY architecture. Section IV proposes an energy consumption evaluation system and reports the evaluation results. Section V targets at the timing of architecture and the zero-skipping ability. Section VI concludes.

II. Motivation

In this section, we introduce state-of-the-art hardware implementation methods of CNN based on exploiting the data reuse of convolution layers and the RS dataflow used in Eyeriss. By analyzing the redundant data access and storage of Eyeriss at the RF level, we motivate COSY's basic thought of shared internal storage.

A. Data Reuse in CNN Hardware Implementation

A general CNN model is composed of several layers, including Conv layers, FC layers, Pooling layers, Non-linearity layers, which are connected as a directed acyclic graph [27]. Conv layers are the most important layers in CNNs, and occupy over 90% of the operations and running time. In general, the Conv layer is calculated as follows:

$$\begin{aligned} O[z][x][y] = &\sum_{k=0}^{C-1}\sum_{i=0}^{R-1}\sum_{j=0}^{R-1} I[k][Sx+i][Sy+j] \\ &\times W[z][k][i][j] + B[z], \\ 0 \le x, y < &\frac{H-R+S}{S}, 0 \le z < M \end{aligned} \tag{1}$$

where O, I, W and B are respectively the matrices of the output feature map, input feature map, filter maps and 1-D bias. H and R mean the size of input feature map and filter matrix. C and M are number of channels and filter matrices. S is the stride of sliding window.

In (1), the huge amount of MAC (multiply-and-accumulate) results in a huge amount of data movements. For its hardware implementation, it is a big challenge to take care of both the operation time and storage access. Usually, the more calculation resource we use to improve the operation efficiency, the larger bandwidth we need to bear data access.

Table I
Normalized energy consumption relative to a MAC operation in a commercial 65nm process

	DRAM	Global Buffer (SRAM)	Array (inter-PE)	PE (0.5kB)
Normalized Energy	200×	6×	2×	1×

In order to utilize the limited bandwidth efficiently and reduce the energy consumption, we need to make good use of data reuse, which is a notable feature of Conv layer. The storage levels, from high to low, are summarized as DRAM, Global Buffer (SRAM), Array and RF, shown in Table I [23]. It is evident that with the raise of the storage level, data access costs more and the demand for top-level bandwidth increases, which improves the energy consumption. Therefore, relocating the data reuse to the lower storage level as much as possible is one of the most effective way to reduce bandwidth utilization and energy consumption.

The previously proposed hardware implementations of CNN usually exploit the data reuse in the following ways:

- **For reuse of filter:** As a sliding window, filter matrices are usually placed adjacent to arithmetic module and accessed in a fixed way to multiply with different image map pixel. The filter reuse usually occurs at the RF level [11, 18, 22]. In addition, a filter matrix can be broadcast, allowing multiple PEs to use it at the same time [28].
- **For reuse of input feature map pixel:** Adopt the register pipeline structure and read regularly through line buffer and shift register, so that different PEs or MACs use the same input pixels at different times [11, 12, 17]. The no redundant data access can also be achieved by using special input pixel arrangement, which is generally accomplished by FIFO [18, 28]. The same as filter matrices, the image pixels is also usally broadcast to multiple PEs or MACs.
- **For reuse of input feature map:** Through broadcasting, the same input feature map is used by multiple PEs at the same time [17, 18, 21]. Typically, there are a plurality of PE groups with the same construction which perform multiple convolution operations in parallel.

Taking all of the reuse patterns into account is a difficult task. Eyeriss [20] makes a good try. It adopts the RS architecture which breaks the high-dimensional convolution down into row-based 1-D convolution primitives. Fig. 1 shows its schematic diagram that consists of R rows and

$(H - R + 1)$ columns of PEs, which performs the 2-D convolution of a H^2 input feature map and a R^2 filter. Before calculation, the data in PEs is configured through multicast buses. Each row of the filter matrix and each row of the input feature map is moved to the corresponding row and diagonal respectively. Each PE executes a 1-D convolution of a filter row and an input feature map row. Each column of the PE array calculates in pipeline, while the $(H-R+1)$ columns of PEs run in parallel, producing partial sums from the buttom to the top of the array.

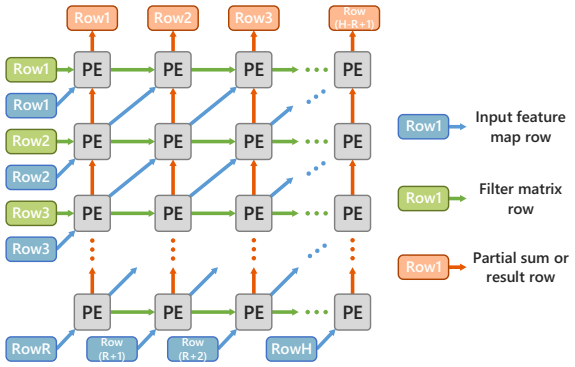


Figure 1. The diagram of the Eyeriss RS dataflow.

The Eyeriss RS dataflow exploits the filter reuse, input feature map pixel reuse and input feature map reuse in convolution operation by all-local storage. The image scratch pad and filter scratch pad inside the PEs transfer most of the data reuse to the RF level.

B. The Redundant Data Access and Storage Usage of Eyeriss at the RF level

Through detailed analysis, we find that there are still redundant storage capacity and data handling in Eyeriss RS dataflow, which brings unnecessary energy consumption:

- **For filter:** In Eyeriss, each PE has its own filter scratch pad. In each row of PE, all the filter scratch pads store the same row of filter and have the same access process. Each value in a filter matrix is accessed $(H-R+S)/S$ times by one PE, and is accessed $((H-R+S)/S)^2$ times by all the $(H-R+S)/S$ PEs. If we find a way to allow each PE of a row to share one filter scratch pad, the total capacity of storage and the total access of RF data will decrease $(H-R+S)/S$ times. For each layer in the CNN, the frequency of the filter matrices access at the RF level will reduce from $(H-R+S)^2/S^2R^2MC$ to $(H-R+S)/SR^2MC$. In AlexNet, the reduction rate can range from 92% to 98%.
- **For input feature map:** Each PE also has its own image scratch pad at the RF level in Eyeriss structure. All PEs on the diagonal line store the same row of the input feature map, and share the same access pattern. Each pixel of the input feature map will be accessed by up to RH times. Therefore, in a complete Conv layer, the feature map pixel access in the RF will happen about RH^3CM times. If we find a way to allow each PE of a diagonal line to share one image scratch pad, the total capacity of storage and the total access of RF data will decrease $R(H-R+S)/SH$ times, and for each layer in the CNN, the frequency of the input feature map pixel access at the RF level will reduce from $(H-R+S)^2/S^2R^2MC$ to $(H-R+S)^2/S^2R^2MC$. In AlexNet, the reduction rate can range from 62% to 77%.

Unfortunately, because of the requirement to adapt to different convolution patterns, the structure of Eyeriss is hard to share image scratch pad and filter scratch pad between PEs. Firstly, the PEs in each row may have different filter data, bringing the complexity of controlling when filter scratch pads are shared. Secondly, for different patterns of convolutions, it is difficult to share the image scratch pad between PEs in a diagonal line. This is because the diagonal directions of different stride values are not the same, and PEs in a diagonal line may have different input feature map values when the PE array implements multiple convolutions using multiple blocks.

Furthermore, if input feature map data is used by PEs in a diagonal line at the same time, a row of filter matrix can not be used by PEs in a row. Therefore, even if we replace the image scratch pad and filter scratch pad with shared storage, it is almost impossible to share the filter scratch pad and image scratch pad simultaneously.

We need to find a way to achieve the data sharing of the two scratch pads on the basis of the Eyeriss RS dataflow, so that the storage capacity and the access frequency at the RF level can be reduced. But note that even if the sharing of the two scratch pads is achieved, the two types of the access frequency reduction can only achieve one, due to the fact that when an input feature map pixel is used by multiple filter matrices at the same time, a pixel in a filter matrix can not multiply by a plurality of input feature map pixels simultaneously.

III. The COSY Architecture

In this section, we adopt the design method of systolic array to compress the Eyeriss RS dataflow to a 1-D array, then get the 2-D array by simply copying it. The proposed architecture is called COSY (CNN on Systolic Array), which achieves the storage sharing between PEs (in RS dataflow) at the RF level as much as possible. It has no redundant input feature map and filter storage, reducing the amount of internal PE storage and the number of the access at the RF level.

A. Derivation: From Eyeriss to COSY

Systolic architectures represent a network of processing elements (PEs) that rhythmically compute and pass data

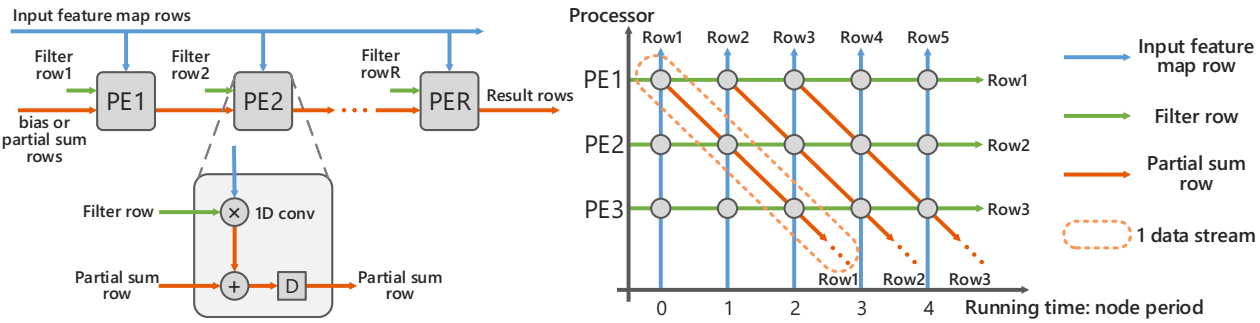


Figure 2. The preliminary model and its space-time representation of the COSY basic array.

through the system. For any given regular iterative algorithm, the corresponding systolic architectures can be designed on using linear mapping techniques on regular dependence graphs (DG). The linear mapping technique maps a regular N-dimensional DG to a lower dimensional systolic architecture [29]. We find that the logic data of Eyeriss can be considered as a regular DG, because all of the nodes have the same 3 direction: input feature map row(to the left), filter row(to the upper right), partial sum(upward), shown in Fig. 1. So we can compress Eyerisss 2-D array into 1-D systolic array using linear systolic mapping methodology. The derivation is given as follows:

1) Select the space representation: We define i and j axis to determine the DG of Eyeriss RS dataflow, which point to the direction of the filter and the partial sum movement respectively. They lead to 3 fundamental edges: input feature map edge $A = (1\ 1)$, filter matrix edge $B = (1\ 0)$, partial sum edge $C = (0\ -1)$. Note that the partial sum edge is not $(0\ 1)$, which is for later convenience.

2) Select the basic vectors of systolic array: Different selections of the basic vectors result in different final results. In order to meet the requirement for storage sharing between PEs, we need to find basic vectors that can make the transmission route of input feature map or filter matrix projected to one node. First of all, define the projection vector $\mathbf{d} = (1\ 0)^T$ and the processor space vector $\mathbf{p}^T = (0\ -1)$, so as not to disrupt the basic structure of RS dataflow. All nodes on a horizontal line will be mapped to the same processor, also making the filter storage of PEs in a row of Eyeriss shared. In addition, any node with coordinate $I = (i\ j)^T$ would be executed at time, $\mathbf{s}^T I$. Therefore, making the scheduling vector $\mathbf{s}$ perpendicular to the transmission direction of input feature map is a good idea to execute computation of PEs of Eyeriss on a diagonal at the same time. So we define the scheduling vector $\mathbf{s} = (1\ -1)^T$. In this step we get the hardware utilization efficiency (*HUE*) of this design, *HUE*$= 1/\mathbf{s}^T\mathbf{d} = 1$.

3) edge mapping: The 3 fundamental edges corresponding to input feature map, filter matrix and partial sum can be mapped to corresponding edges in the systolic array. If an edge $\mathbf{e}$ exists in the space representation or DG, then an edge $\mathbf{p}^T\mathbf{e}$ is introduced in the systolic array with $\mathbf{s}^T\mathbf{e}$ delays. The mapping result is shown in table II and Fig. 3.

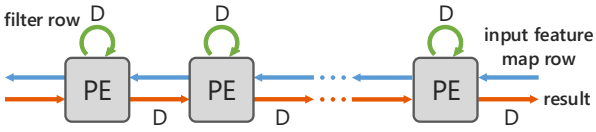


Figure 3. The block diagram of the systolic array of the RS dataflow.

Table II
THE CONSTRUCTION PARAMETERS OF THE SYSTOLIC ARRAY

$\mathbf{e}^T$	$\mathbf{p}^T\mathbf{e}$	$\mathbf{s}^T\mathbf{e}$
$A(1\ 1)$	-1	0
$B(1\ 0)$	0	1
$C(1\ 1)$	1	1

4) Hardware mapping: This step maps the block diagram Fig. 3 to hardware roughly. Firstly, in Fig. 3, there is no delay in the transmission path of input feature map, which means the row of input data is broadcast to the PEs. Secondly, the filter matrix values stay in their own processors, and transfer the same row to compute each time. Note that the different rows of filter matrix are respectively stored in different PEs. Third, a PE in the array performs a 1-D convolution operation, which is the same as the PE in Eyeriss. Finally, the 1 cycle delay in Fig. 3 means transferring partial sum to the next PE for accumulation in the next calculation cycle of PE. We call 1 calculation cycle/transfer delay as 1 node period. The preliminary model and its space-time representation is shown in Fig. 2.

Now we get the basic array of COSY, which is able to calculate a 2-D convolution with R^2 filter using R PEs. All of the PEs share the same input feature map through broadcast from one shared storage and no redundant storage of filter exists. Therefore, the problems mentioned in Section III are solved preliminarily.

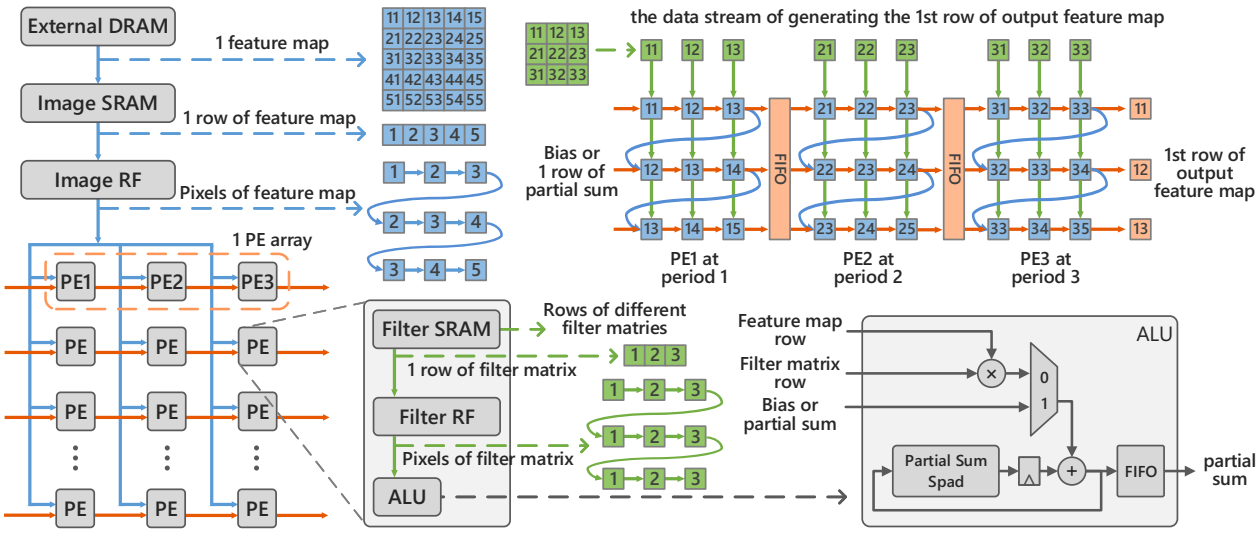


Figure 4. The overall COSY architecture including the different storage levels and the basic internal structure of PE. It also shows the data stream of generating the first row of 2-D convolution result by 3 PEs, in the case of 5×5 input feature map and 3×3 filter matrix.

B. Core Architecture and Dataflow of COSY

COSY's core architecture consists of multiple parallel basic arrays. The detail of the architecture and dataflow COSY are introduced as follows, from the aspect of data loading, partial sum calculation and data transmission between PEs:

1) Loading of input feature map: In a systolic array of COSY, the same input feature map row is broadcast to all the PEs. Therefore, multi-channel PE arrays run in parallel, and get multiple 2-D convolution results of one input feature map and multiple filter matrices. The input feature map is transferred from an external DRAM to a SRAM that can accommodate one 2-D feature map, then transferred to the RF block row by row. The RF block stores only one row of feature map, and the address pointer fetches data in it once per cycle. Then the feature map pixels are transferred to each PE. Unlike the single PE array of Eyeriss, there is no image scratch pad that stores the input feature map in the PE of COSY, thus our PE can be maintained in a smaller area and can be close to the RF as much as possible.

2) Loading of filter matrices: In each PE, there is a filter memory that has the same function as the Eyerisss filter scratch pad, which is used to transfer the filter matrix to the arithmetic logic unit (ALU). The SRAM in the PE prestores filter data before the calculation and transfers the data row by row to the RF. Then the RF outputs the data to the ALU during the calculation cyclically.

3) Calculation of Partial Sum and data transfer between PEs: The internal structure of the COSY's PE is similar to that of Eyeriss, except that the image scratch pad is removed. The PE completes a 1-D convolution in $R(H-R+S)/S$ cycles (1 node period), and places the resulting partial sum vector in a small FIFO. The vector is then transferred to the next PE or stored as a row of the 2-D convolution result. Each PE has an input sum interface used to receive the partial sum of the previous PE or the bias of this layer.

As we can see in Fig. 2, the systolic array runs in a full pipelined way. During most of the time, all the R PEs execute the same form 1-D convolution synchronously. To calculate a complete 2-D convolution, $(H-R+1)$ data streams shown in Fig.2 are needed. When a 2-D convolution calculation is complete, the next one can be performed. Since multiple PE arrays share the same input feature map in RF, multiple 2-D convolutions can be executed simultaneously with little input bandwidth occupied. Fig. 4 shows the overall COSY architecture, including the different storage levels and the basic internal structure of PE. It also shows how the 3 PEs get the first row of 2-D convolution result, in the case of 5×5 input feature map and 3×3 filter matrix.

C. Configurability and Control Complexity

The configurability suitable for different convolutional parameters is the feature that the COSY must satisfy, which is demonstrated as follows:

1) Different size of input feature map: It can be achieved by broadcasting feature map row to PEs with the corresponding length.

2) Different size of filter matrix: We can simply prestore the data to the filter SRAM regularly and transfer the data to the ALU cyclically by the size of filter matrix row. The number of the PE in an array is fixed to 3, because the filter size in most of the latest popular CNN structures is 3×3. Besides, there are also some filter matrix sizes like 5×5 and 11×11 appearing in some CNNs [1], which is solved by stitching (Fig. 5). It can be understood as follows: the space-time representation is divided horizontally into

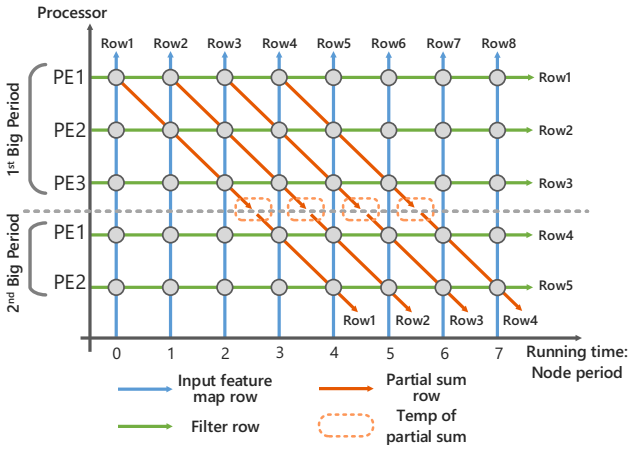


Figure 5. Use stitching to calculate the 2-D convolution when the filter size R is larger than the number of the PE in a array. In this example, $R = 5$ and the number of the PE in a array is 3.

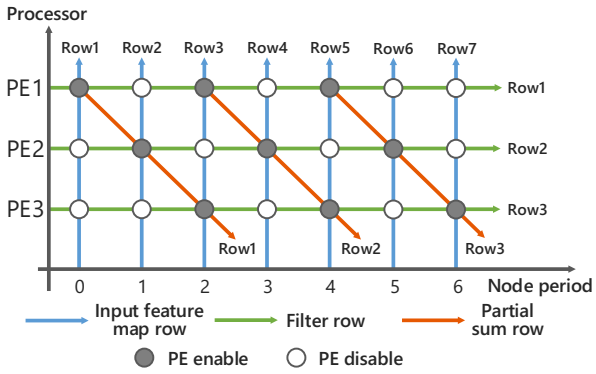


Figure 6. An example showing how we achieve the convolution of $S > 1$ by turn PE on and off regularly. In this example, $S = 2$ and $R = 3$. The big period means the different part of the complete 2-D or 3-D convolution.

multiple parts, each can be implemented in existing PE array completely. The complete 2-D convolution is achieved by stitching these parts of the operations together. In addition, the intermediate value of each part must be stored in a dedicated storage as the partial sum input of the next part.

3) Different stride: Conv layers with stride values larger than 1 [1] exist in some CNN models. The size configurability can be achieved by turning PE on and off regularly. As shown in Fig. 6, if we turn the PE on every S node periods, the convolution with the stride value S is achieved. The status of each PE depends on whether the previous PE is on in the previous node period, so the enable signal is transferred within the PE array along with the data stream. Therefore, only the first PE in an array is needed to control. We can get the hardware utilization $HUE = 1/S$, which is much lower. Fortunately, the energy consumption will not be wasted, since the enable signal is used to turn off the PEs.

In summary, the proposed architecture COSY is applicable to a variety of convolutional parameters, suitable to different kinds of Conv layers. It also shows a low control complexity.

Relatively speaking, Eyeriss is more complex in controlling because all the PEs needs to be configured for different modes before calculation. In COSY, the data preparation and the calculation is able to be done at the same time, and the control signals (such as enable signal) is also pipelined along with the data. If we use a central control module, it can connect to PEs via fewer connections.

D. An Example of Conv Layer Computation

In order to describe the process of the Conv layer computation on COSY more clearly, we give an example. The Fig. 7 demonstrates the usage of each row of the filter matrix and the dataflow of the intermediate cache in one Conv layer of CNN, when the filter size R is 5, the number of input feature map channels C is 2, the filer number M is 4, the PE number per an array (n) is 3, and the number of the PE arrays (m) is 2. abc represents the cth row of the bth channel in the ath 3-D convolution. Since two PE arrays share the same input feature map, four 2-D convolutions in one channel will calculate in two to complete. A 2-D convolution can not complete using one pipeline calculation, and must be divided into two systolic pipelines. The partial sum that cannot be transferred to PE in the current pipeline directly is cached in the SRAM as $temp$. A 3-D convolution is obtained by adding two channels of 2-D convolutions, and the first 3-D convolution result is cached in SRAM as $tempout$. In the later operations, $temp$s and $tempout$s are transferred into the first PE of the PE arrays in the same way as bias. The bias data is input into the PE array in the first 2-D convolution period of each channel. The final results of the four 3-D convolution operations are stored in DRAM.

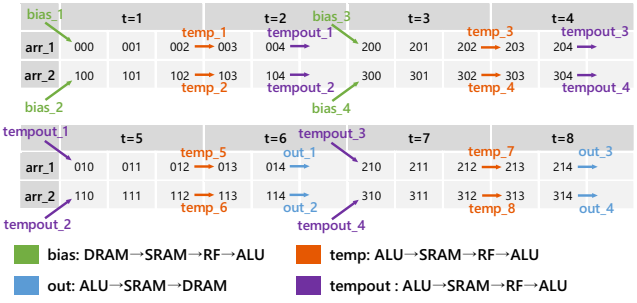
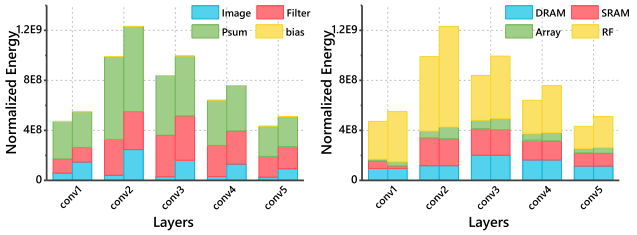


Figure 7. An example demonstrating the usage of each row of the filter matrix and the dataflow of the intermediate cache in one Conv layer of CNN. In this example, $R = 5$, $C = 2$, $M = 4$, $n = 3$, $m = 2$. The t value represents the big period number in the Conv layer.

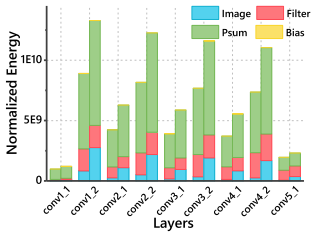
IV. Energy Consumption Evaluation System

In this section, we develop an evaluation system to estimate the energy consumption of COSY in case of deploying different CNN models on different hardware platforms as

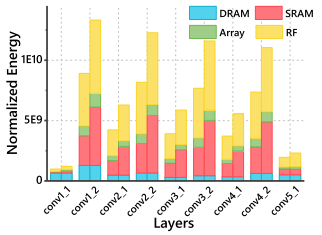


(a) AlexNet, 4 data types

(b) AlexNet, 4 data access levels

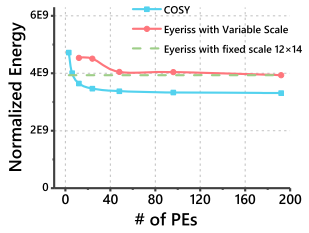


(c) VGG16, 4 data types

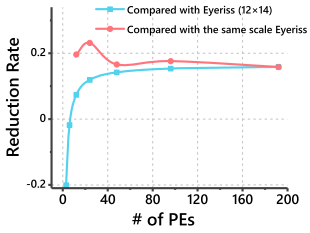


(d) VGG16, 4 data access levels

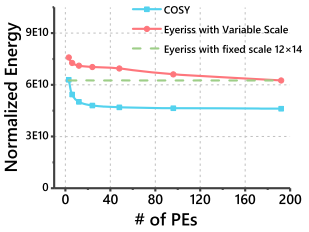
Figure 8. The normalized energy consumption of the different layers of (a)(b) AlexNet and (c)(d) VGG16 implemented on COSY and Eyeriss. (a) and (c) are classified into 4 data types, (b) and (d) are classified into 4 data access levels. The COSY architecture has 16 PE arrays with 3 PEs in an array. The Eyeriss has 4 columns and 12 rows in (a) and (b), while has 6 column and 3 rows in (c) and (d).



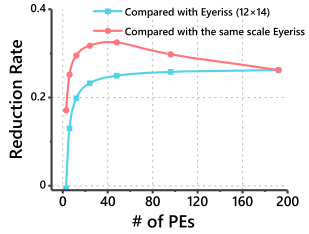
(a) AlexNet, total energy



(b) AlexNet, reduction rate



(c) VGG16, total energy



(d) VGG16, reduction rate

Figure 9. The total normalized energy consumption of (a)AlexNet and (c)VGG16 on COSY and Eyeriss, and the energy reduction rate of (b)AlexNet and (d)VGG16 on COSY compared with Eyeriss.

accurately as possible without real implementation. All of the results are gotten compared with Eyeriss RS dataflow, since other types of architecture are discussed in [23].

A. A Process-based Energy Consumption Estimation System

[23] proposed the method that estimates the energy consumption based on data movement hierarchy. The energy consumption of calculation can be estimated without real implementation, because the dataflow is definite when the CNN model and hardware architecture are given certainly. The proposed system aims to evaluate and compare the energy consumption of COSY and Eyeriss by specifying an arbitrary CNN network structure and the hardware scale of COSY and Eyeriss. The result is obtained from the actual process of operations and dataflow.

Firstly, we use the CNN model defined in Caffe framework [30] to obtain the parameters. Secondly, we analyze the data handling mode of COSY and Eyeriss by simulating the actual operations. Then we accumulate all the storage access times by categories, including 4 storage levels (*input feature map* (*image*), *filter matrix*, *partial sum* (*psum*, including *out*, *temp*, *tempout*), *Bias*) and 4 data types (*DRAM*, *SRAM*, *Array*, *RF*). Finally, we calculate the energy consumption of these categories and sum them according to Table I. The table shows the normalized energy consumption relative to a MAC operation extracted from a commercial 65nm process. These storage levels of normalized energy consumption are similar in other processes or platforms. We built the system upon the Matlab platfrom.

B. Results

We evaluate the energy consumption of two typical CNN models, AlexNet and VGG16, on COSY and Eyeriss RS dataflow. We analyze their energy consumptions on different hardware scales, and prove the energy reduction in 8 aspects mentioned in Section IV-A.

1) Normalized Energy consumptions on COSY and Eyeriss with fixed scale: Since COSY's and Eyeriss's PEs have a similar form, it is reasonable to evaluate and compare these architecture with the same PE number. In COSY, we fix the PE scale to 3×16, which means there are 16 PE arrays with 3 PEs in an array. Correspondingly, the Eyeriss architecture to be tested has 48 PEs. For VGG16, we let the scale of Eyeriss 16 columns and 3 rows since the filter size 3, while for AlexNet these are 4 columns and 12 rows because we want to fix the number of rows the same as the actual Eyeriss architecture. Fig. 8 gives the normalized energy consumption of different Conv layers of AlexNet and VGG16 simulated in the aforementioned conditions. The figure shows that COSY architecture achieves an impressive reduction compared with Eyeriss. For AlexNet, the reduction rate of different layers ranges from 14.1% to 19.6% and total reduction rate is 16.6%. For VGG16 it is 32.44% in total. In terms of data types, the reduction mainly comes from the energy reduction of input feature map (over than 75% in total), while the costs of other 3 types are much less reduced. In terms of storage levels, the data access energy at the SRAM, the array and the RF level reduce to varying degrees. For AlexNet, the reduction mainly comes from the RF level, and for VGG16, it mainly comes from the RF level and the SRAM level. The

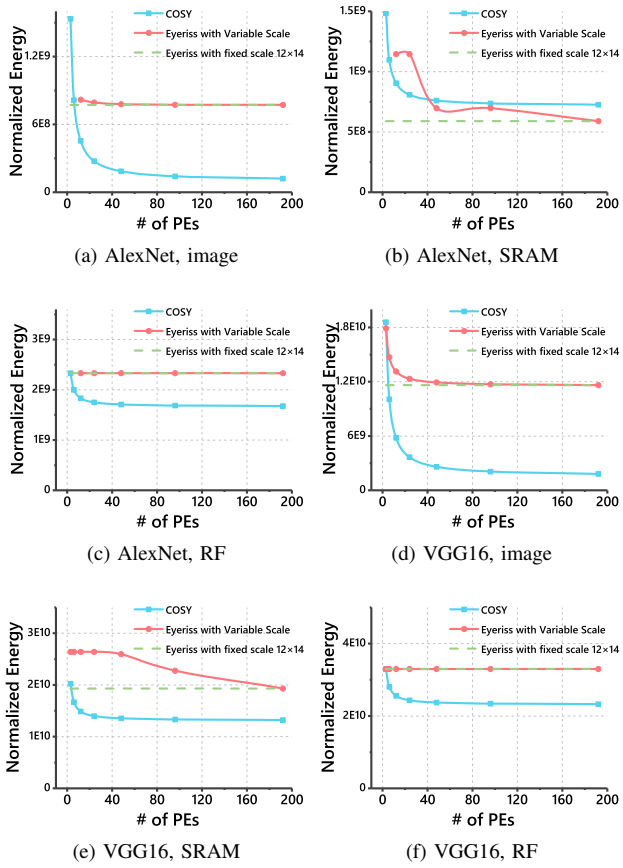


Figure 10. Three aspects of the total normalized energy consumption of Conv layers in (a)(b)(c) AlexNet and (d)(e)(f) VGG16 implemented on COSY and Eyeriss with the variable scale. (a)(d) show the normalized energy consumption of transferring input feature map (image). (b)(e) show the data access energy at the SRAM level. (c)(f) show the data access energy at the RF level. The horizontal ordinate of each figure is the total number of PEs. They are also compared with the actual Eyeriss architecture (12×14).

data access at the DRAM level in COSY is the same as that in Eyeriss.

2) Normalized Energy consumptions on COSY and Eyeriss with variable scale: For COSY, we fix the number of PEs in an array to be 3 and change the number of PE arrays. For VGG16 on Eyeriss, we fix the number of rows to 3 and change the number of columns. For AlexNet on Eyeriss, we fix the number of rows to 12. The total number of PEs in Eyeriss and COSY keeps the same. Fig. 9 shows the total normalized energy consumption of AlexNet and VGG16 on COSY and Eyeriss, and the energy reduction rate of AlexNet and VGG16 on COSY compared with Eyeriss. We see from the figure that with the increase in the number of PEs, the energy on COSY and Eyeriss is reducing and eventually closes to a fixed value. The energy of COSY becomes less than Eyeriss when the PE number is bigger than 6. For AlexNet, COSY costs around 15% less than Eyeriss with the same hardware scale. For VGG16, the reduction rate is more than 20%, and when they have 48 PEs, the value reaches the maximum number 32%. 48 PEs is the most reasonable scale for COSY, since they achieve a low energy consumption using smallest scale. For more detailed analysis, 3 aspects of energy consumptions (*input feature map* (*image*), *the data access at the SRAM* and *the RF level*) are counted and displayed in Fig. 10, because the energy reduction of COSY is mainly comes from them, and the other 5 aspects have an insignificant impact. We also compare them with the actual Eyeriss architecture (12×14). The results show that the transfer cost of input feature map (image) on COSY is rapidly decreasing with the increase in the number of PEs, and finally converges to less than 20% of that on Eyeriss. This reduction mainly comes from the storage sharing of image scratch pad between PEs at the RF level. The more the number of PE arrays, the less the data access times of the input feature map at the RF level and the intermediate cache at the SRAM level. For AlexNet, the energy cost existing at the SRAM level on COSY is larger, which is different from that of VGG16, because the COSY architecture has only 3 PEs in an array so that more cache is needed for over 3×3 filter matrix in AlexNet.

In summary, the COSY architecture shows an advantage in the aspect of energy consumption when shares the same scale with Eyeriss, proving that the sharing of image and filter scratch pad between PEs by adapting systolic array can effectively reduce the energy cost of data access at the RF level, especially for input feature map.

V. Timing Analysis: The Intrinsic Zero-skipping Ability

This section targets at timing analysis. We prove that the proposed architecture spends almost the same computing time to execute convolution operations as Eyeriss with the same scale when the stride value $S = 1$. We also find that COSY has the intrinsic zero-skipping ability while the Eyeriss has not, which further improves the efficiency of the architecture.

A. Timing Analysis Compared with Eyeriss

Fig. 11 shows the rough timing diagram of 2-D convolution on COSY and Eyeriss in the case of $H = 5$ and $R = 3$. In Eyeriss, it takes 15 cycles to generate the first row of output feature map. Meanwhile, three columns of PEs run in parallel to generate the all 3 rows of output matrix, so that only 15 cycles is needed for an entire 2-D convolution. As for COSY, it takes 45 cycles to get a 3×3 output matrix by 3 PEs in a PE array. However, 3 PE arrays can run in parallel. If we want to calculate $(H - R + S)/S$ 2-D convolutions, $(H - R + S)/S$ PE arrays that have R PEs in a array are used in COSY, taking $HR(H - R + S)/S$ cycles. If we use Eyeriss with $(H - R + S)/S$ columns and R rows, it takes $HR(H - R + S/S)(H - R + RS)/S$ cycles.

When $S = 1$, the numbers of cycles are equal. For $S > 1$, COSY spends more time than Eyeriss. Fortunately, most of the Conv layers of popular CNNs have a stride value of 1, and the layers with a value of over 1 only occupies a small amount of computation.

In conclusion, COSY takes almost the same computing time of convolution as Eyeriss with the same hardware scale.

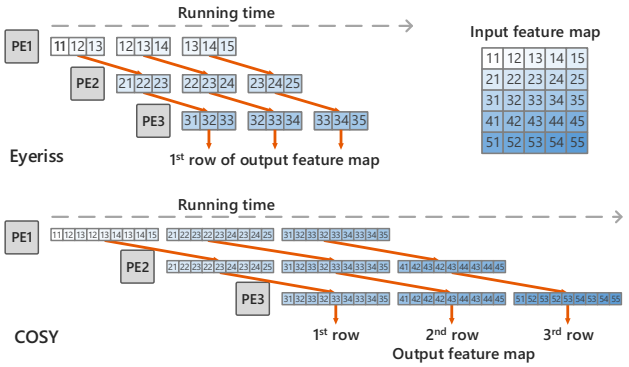


Figure 11. A rough timing diagram of 2-D convolution on COSY and Eyeriss, in the case of $H = 5$ and $R = 3$. In Eyeriss's timing diagram, we only show how to generate the first row of output feature map.

B. The Intrinsic Zero-skipping Ability

In CNNs, many of the feature map pixels turn out to be zeros. For state-of-the-art CNNs, the average total fraction of the pixels that are zeros varies from 37% to up to 50%, and the average value is 44% [19]. These zero values do not contribute to the final result, which need to be removed and skipped over.

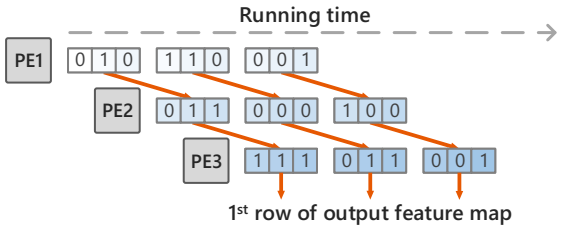


Figure 12. The rough timing diagram of 2-D convolution on Eyeriss, which shows why it can not skip over zero value in timing. In this example, $H = 5$ and $R = 3$. The value 1 represents a non-zero value.

Fig. 12 shows why Eyeriss can not skip over zero in timing. If we skip over zeros in one of the PEs, the other PEs will usually still need to wait certain cycles and spend certain time to execute the multiplication and accumulation, unless all the PEs get the zero value at the same time, which is hard to meet. In other words, the input data and clock are strictly bound. Therefore, Eyeriss architecture does not support for zero-skipping intrinsically.

COSY has solved the problem and has the ability of intrinsic zero-skipping, due to the input broadcast mechanism. As shown in Fig. 13, when the pixel of input feature map is zero, all the PEs in the entire structure can skip over synchronously without affecting the timing of data transmission. Similar to Cnvlutin, before computing, rows of input feature map are sorted into non-zero arrays, and each pixel is bounded to the corresponding filter pixel address. If the input values have R zeros continuously, the corresponding pixel of new array is set to be zero, leaving one cycle for the PEs to output partial sum.

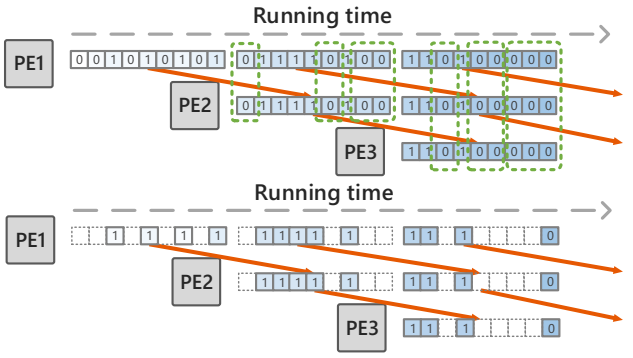


Figure 13. A rough timing diagram of 2-D convolution on COSY, which shows how it skips zero value. In this example, $H = 5$ and $R = 3$. The value 1 represents a non-zero value.

C. Efficiency Improvements

The Energy-Delay Product (*EDP*) [25] and Energy-Delay Squared Product (ED^2P) [26] are two commonly used metrics to compare two computing architectures considering energy and performance. In section V-C, the average reduction rate of COSY is around 25%, so the *EDP* and ED^2P improvements are 1.33 on average without zero-skipping. Due to the intrinsic ability of zero-skipping, the operation time of COSY can be well compressed. Since the average 44% of the input pixels are zeros, the COSY reduce computing time by 40.5%, according to our simulation. Therefore, the *EDP* and ED^2P of COSY with zero-skipping improves by 2.25× and 3.83× respectively.

In addition, pruning method which removes ineffectual neurons (feature map pixels) [31–33] can reduce the zero value rate and further compress the computing time, and improve the *EDP* and ED^2P value. The improvement depends on the threshold of pruning.

VI. Conclusion

The purpose of this work is to develop an energy-efficient hardware architecture executing deep convolutional neural networks (CNNs) suitable for resource-constrained devices.

This work aims at Conv layers in CNN algorithm. By analyzing the data reuse mode of CNN on hardware, especially the redundant data access and storage usage of Eyeriss at the RF level, this work adopts the design method of systolic array to achieve the storage sharing between PEs, exploiting the data access at the RF level.

The proposed architecture COSY shows an advantage in the aspect of energy consumption when shares the same scale with Eyeriss. It achieves an over 15% reduction in energy consumption under the same hardware and algorithm constraints, improving theoretical *EDP* and ED^2P by 1.33× compared with Eyeriss. It is also proved to have intrinsic support for zero-skipping, which can further increase the improvements to 2.25× and 3.83×.

ACKNOWLEDGMENT

We thank the anonymous reviewers for their comments and suggestions. This work is supported by R&D Project of Shenzhen Government (Project JCYJ20160229094148396 and JCYJ20160428153956266).

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NIPS*, 2012.

[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *Computer Science*, 2014.

[3] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *IEEE CVPR*, 2015.

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE CVPR*, 2016.

[5] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva, "Learning deep features for scene recognition using places database," in *NIPS*, 2014.

[6] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, 2015.

[7] J. Cong and B. Xiao, "Minimizing computation in convolutional neural networks," in *ICANN*, 2014.

[8] J. Zhang and J. Li, "Improving the performance of opencl-based fpga accelerator for convolutional neural network," in *FPGA*, 2017.

[9] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing fpga-based accelerator design for deep convolutional neural networks," in *FPGA*, 2015.

[10] N. Suda, V. Chandra, G. Dasika, A. Mohanty, Y. Ma, S. Vrudhula, J.-s. Seo, and Y. Cao, "Throughput-optimized opencl-based fpga accelerator for large-scale convolutional neural networks," in *FGPA*, 2016.

[11] M. Sankaradas, V. Jakkula, S. Cadambi, S. Chakradhar, I. Durdanovic, E. Cosatto, and H. P. Graf, "A massively parallel coprocessor for convolutional neural networks," in *IEEE ASAP*, 2009.

[12] H. Li, X. Fan, L. Jiao, W. Cao, X. Zhou, and L. Wang, "A high performance fpga-based accelerator for large-scale convolutional neural networks," in *FPL*, 2016.

[13] R. DiCecco, G. Lacey, J. Vasiljevic, P. Chow, G. Taylor, and S. Areibi, "Caffeinated fpgas: Fpga framework for convolutional neural networks," *arXiv preprint arXiv:1609.09671*, 2016.

[14] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song *et al.*, "Going deeper with embedded fpga platform for convolutional neural network," in *FPGA*, 2016.

[15] C. Wang, L. Gong, Q. Yu, X. Li, Y. Xie, and X. Zhou, "Dlau: A scalable deep learning accelerator unit on fpga," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 3, 2017.

[16] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: efficient inference engine on compressed deep neural network," in *ISCA*, 2016.

[17] J. Sim, J.-S. Park, M. Kim, D. Bae, Y. Choi, and L.-S. Kim, "14.6 a 1.42 tops/w deep convolutional neural network recognition processor for intelligent ioe systems," in *IEEE ISSCC*, 2016.

[18] L. Cavigelli, D. Gschwend, C. Mayer, S. Willi, B. Muheim, and L. Benini, "Origami: A convolutional network accelerator," in *VLSI*, 2015.

[19] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. E. Jerger, and A. Moshovos, "Cnvlutin: Ineffectual-neuron-free deep neural network computing," in *ISCA*, 2016.

[20] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, 2017.

[21] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun *et al.*, "Dadiannao: A machine-learning supercomputer," in *MICRO*, 2014.

[22] H. Yoo, S. Park, K. Bong, D. Shin, J. Lee, and S. Choi, "A 1.93 tops/w scalable deep learning/inference processor with tetra-parallel mimd architecture for big data applications," in *IEEE ISSCC*, 2015.

[23] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *ISCA*, 2016.

[24] R. Andri, L. Cavigelli, D. Rossi, and L. Benini, "Yodann: An architecture for ultra-low power binary-weight cnn acceleration," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2017.

[25] R. Gonzalez and M. Horowitz, "Energy dissipation in general purpose microprocessors," *IEEE Journal of solid-state circuits*, vol. 31, no. 9, 1996.

[26] A. J. Martin, M. Nyström, and P. I. Pénzes, "Et2: A metric for time and energy efficiency of computation," *Series in Computer Science*, 2002.

[27] Y. LeCun, K. Kavukcuoglu, and C. Farabet, "Convolutional networks and applications in vision," in *IEEE ISCAS*, 2010.

[28] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, "Shidiannao: Shifting vision processing closer to the sensor," in *ACM SIGARCH Computer Architecture News*, vol. 43, no. 3, 2015.

[29] K. K. Parhi, *VLSI digital signal processing systems: design and implementation*. John Wiley & Sons, 2007.

[30] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *ACM Multimedia*, 2014.

[31] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *NIPS*, 1990.

[32] B. Hassibi, D. G. Stork, and G. J. Wolff, "Optimal brain surgeon and general network pruning," in *Neural Networks, 1993., IEEE International Conference on*, 1993.

[33] 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*, 2015.