Low-power Hardware Accelerator for Sparse Matrix Convolution in Deep Neural Network

Erik Anzalone, Maurizio Capra, Riccardo Peloso, Maurizio Martina, and Guido Masera

Politecnico di Torino, Torino TO 10129, Italy,
name.surname@polito.it

Abstract. Deep Neural Networks (DNN) have reached an outstanding accuracy in the past years, often going beyond human abilities. Nowadays, DNNs are widely used in many Artificial Intelligence (AI) applications such as computer vision, natural language processing and autonomous driving. However, these incredible performance come at a high computational cost, requiring complex hardware platforms. Therefore, the need for dedicated hardware accelerators able to drastically speed up the execution by preserving a low-power attitude arise. This paper presents innovative techniques able to tackle matrix sparsity in convolutional DNNs due to non-linear activation functions. Developed architectures allow to skip unnecessary operations, like zero multiplications, without sacrificing accuracy or throughput and improving the energy efficiency. Such improvement could enhance the performance of embedded limited-budget battery applications, where cost-effective hardware, accuracy and duration are critical to expanding the deployment of AI.

Keywords: Deep Learning, Deep Neural Network, Machine Learning, Energy-efficient Hardware Accelerator, Low-power, ASIC, VLSI

1 Introduction

Artificial Intelligence (AI) has definitely become an undeniable part of human life. The increasing interest the research world is addressing towards this topic is the result of the astonishing performance of the AI approach. Scientists look at the AI as a playground, where they get to reproduce human brain behaviors such as reasoning, learning and problem-solving. AI will soon permeate every aspect of our lives, changing the way we perceive the world, offering new tools able to assist people both during work time and in their everyday life. In fact, many are the rising applications aimed at improving the quality of the work such as the medical environment [1], where, for example, aiding software have been designed to refine specialist x-rays scans interpretation.
Even though the state of the art is far from producing devices with intrinsic manlike properties of reasoning and creativity, recent developments in brain-inspired algorithms have enormously enhanced computers ability in classification tasks [2–6], going beyond human accuracy, leading the emergence of the Machine

Learning (ML) and its narrower area, named Deep Learning (DL) [7].
In recent years, DL gained visibility due to its potential in speech recognition, computer vision and robotics. Such models rely on Neural Network (NN), or more specifically on Deep Neural Network (DNN) [8], brain-inspired structures composed of several layers of artificial neurons, that represent the fundamental building blocks. These blocks are a mathematical representation of the physic principle of how a neuron is supposed to work. Basically, a multiply and accumulate (MAC) operation, followed by a non-linear activation function.
Thanks to subsequent layers of neurons, DNNs are able to extract high-level features from raw data, making them accuracy superior to any other known approach. The most effective DNNs used today, especially in computer vision, are the convolutional ones, where the basic task performed by each layer is the convolution between a feature map and a filter, also called kernel. Even though such operations are not complex, they come in a huge amount, making the above structures very computation-hungry and, subsequently, power-hungry.
DNNs can work in two separated phases, namely training and inference. During the former, the network tries to learn and generalize a given labeled dataset by tuning its weights (kernels), while in the latter it exploits the trained weights to predict and classify the input.
In the above-mentioned scenario, where DL energy greedy applications are becoming more and more pervasive, the need for tailored low-power hardware platforms arises [8]. Whilst during the training stage, parallel computation and high precision (floating point architecture) are required, making the GPU (Graphics Processing Unit) the only possible choice in order to speed up the process, such constraints are not imperative for inference, allowing for a different approach. CPU represents a reasonable choice, but its general purpose design appears not optimized enough to reach high-efficiency levels. Therefore, two hardware solutions exist able to lead DNNs towards a low-power implementation such as FPGAs and ASICs. Although FPGAs allow for greater flexibility by means of their built-in re-programmable property, their power consumption is still too high compared to what ad-hoc integrated solutions are able to achieve. This is the reason why many hardware accelerators for DNNs have been designed as ASIC to be integrated into more complex architectures [8].
From the hardware point of view, two are the main sources of power consumption, namely memory access and multiplication operations. The former is the most energy-expensive, generally tackled at algorithm level by exploiting reutilization policies (introduced in section 2) in order to ward off memory readings. The latter, instead, can be handled by approximate computing, or more effectively, by preventing useless operations like multiplications by zero. This current work is based on the design of a hardware accelerator for convolutional DNN that adopts a rescheduled data flow in order to fulfill the maximum weights reuse. Moreover, algorithmic low-power techniques are applied in order to avoid unnecessary or negligible multiplication between activations and weights.
The paper is organized as follows. Section 2 describes the basic architecture of the hardware accelerator for the convolutional task, including reutilization poli-

cies and rescheduled data flow. Section 3 introduces the low-power techniques developed in this work, with emphasis on the architectural implementations and driving motivations. Section 4 illustrates the results obtained, focusing not only on the outcome of hardware synthesis and validation, but also on comparisons with other known architectures. Section 5 draws the conclusions and possible future works.

2 Basic hardware accelerator

The core structure of a typical convolution accelerator is the MAC (Multiply and Accumulate) unit, as the convolution operation is usually performed as the sum of the products between the convolution kernel coefficients and the corresponding input samples. Eq. 1 refers to a typical discrete convolution adopted in deep learning, with dimensionality equal to 2.

$$\text{output}(x, y) = \sum_{\substack{\text{row}\\\text{column}}} \text{kernel}[r, c] \cdot \text{input}[x + r, y + c] \tag{1}$$

where r and c represent the dimension of the kernel, while x and y are the dimensions of output feature map. There are various possible implementations of this equation in hardware, by interchanging the order of summations and by deciding the flow of data. In particular, there are three possible architectural implementations for fetching and transferring input and weight data from memories to the processing elements (PEs) [9]:

- **Broadcast**, where data are fetched one by one and they are sent to multiple PEs every clock cycle. Therefore, in the same clock cycle each PE receives the same data. Despite one input memory is required, some registers can be used to separate the data flow between memory and PEs;
- **Forwarding**, where data are fetched one by one from the input memory but each PE receives them in different clock cycles, i.e. each PE contains the logic to process the data and some registers to store them. Thus, the data flowing from one PE to the next one are delayed by one clock cycle;
- **Stay**, where data, once loaded inside a PE, are kept fixed for the entire convolution. This is a good way to reduce the number of memory accesses as data are reused by the same PE.

There are also three architectural possibilities to classify the process of partial-sum accumulation:

- **Aggregation**, where all partial-sums are added at the same time by means of a tree adder. Nowadays this solution is the least used one due to the relevant complexity which would be required a parallel structure;
- **Migration**, where the partial sum is transferred to a neighboring PE or to the same PE that generated it;
- **Sedimentation**, where each partial sum is stored into an on-chip memory for each PE.

In this work, a FSM (*Forwarding-Stay-Migration*) approach is used because it is well suited to support low power techniques, as described in section 3. The *Forwarding* approach, applied to inputs loading, allows a delayed parallelization of all the computations. This ensures that input is fetched only once from the input memory. Since the kernel does not change during a convolutional operation, the *Stay* approach is an optimum choice, as kernel weights are loaded just once and remain inside architecture during the entire operation. Eventually, *Migration* has been chosen for the output process, since it allows to use internal registers to transfer the partial sum between two PEs, thus reducing memory operations.

The data input memory (herein called "activation memory"), the weight memory, the output memory and the memory controller are assumed to be off-chip (as in [10] and [11]) to focus on the the hardware accelerator itself, composed by a 3-channels datapath and its control unit. Differently from *Eyeriss*[10], where each PE receives inputs from the same row, and from *Zena*[11], where a zig-zag access pattern is required, in this work a column-by-column access is required for the *Forwarding* input processing. Input data is represented on 9 bits per color, hence 27 bits per pixel, with a resolution of 128x128 pixels, leading to an activation memory of 16384 rows and 27 columns. Due to the *Stay* approach, all the kernel is needed at the same clock cycle. Weights are quantized on 6 bits per color, leading to 162 bits of memory for a 3x3x3 convolution kernel. The output memory is composed by 126x126 rows of 3x16 bits each.

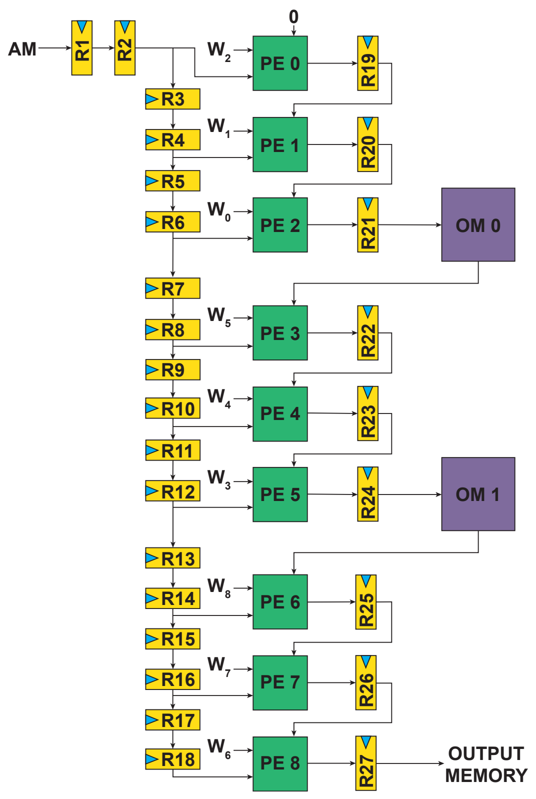
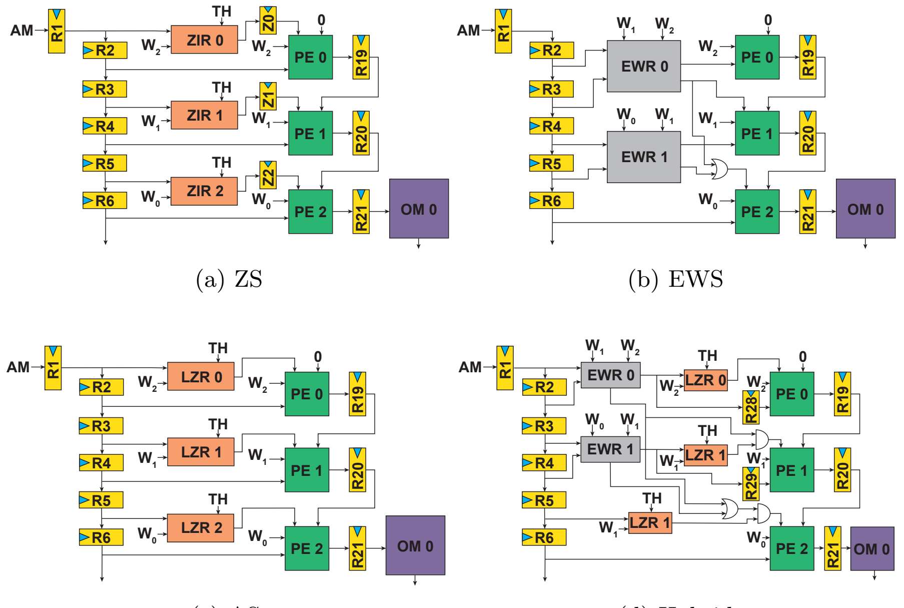


Fig. 1: Basic architecture for *FSM* approach

The Processing Elements, the internal registers and the On-chip memories are embedded in the structure, as shown in figure 1. Using the rescheduled dataflow from [9], a series of registers is inserted between the inputs of the Datapath and the various PEs accordingly to the kernel size, in this case 2 registers to cope with a 3x3 kernel.

3 Power consumption reduction

Rescheduled dataflow is an effective technique to reduce power consumption due to on-chip memory access. Besides, the multiplication is known to be another important source of power consumption. As a consequence, techniques aimed at reducing the power consumption of multipliers are worth studying. For instance, activations and weights can be analyzed to understand which operations can be skipped to create low power architectures. For this reason, in the next sections, four different approaches applied to the starting architecture are presented: Zero Skipping (ZS) Architecture, Equal Weights Skipping (EWS) Architecture, Approximation Skipping (AS) Architecture and Hybrid EWS and AS Architecture.



(a) ZS (b) EWS

(c) AS (d) Hybrid

Fig. 2: Low power optimizations (partial view)

3.1 Zero Skipping

This method allows to skip zero-output operations caused by zero weights or zero activations to reduce energy consumption. In particular this technique is very effective in networks employing the ReLU activation function, which outputs a large number of zeros. The new architecture needs a block named *Zero Input Recognizer* (ZIR) able to detect if at least one operand is zero-valued, skipping the multiplier, as shown in Figure 2a. If a zero is detected by the ZIR, a zero-flag is sent to a modified version of PE capable of skipping the operation.

3.2 Equal Weight Skipping

If two consecutive weights are equal, a multiplication can be traded for an addition as

$$PartialResult = A_2W_2 + A_1W_1 + A_0W_0 \tag{2}$$

with $W_2 = W_1$ can be rewritten as

$$PartialResult = (A_2 + A_1)W_1 + A_0W_0 \tag{3}$$

which requires one less multiplication. This optimization requires a new block called *Equal Weights Recognizer* (EWR) which looks for equal weights, as shown in Figure 2b. This results to be a low-power optimization thanks to the *Stay* approach.

3.3 Approximation Skipping

The third proposed low power technique consists of an approximation of the convolution operation. It is possible to skip multiplications which results are known to be negligible by counting the number of operands *leading zeros*. The multiplication returns a number having as much *leading zeros* as the sum of the ones in the input. A threshold can be set to avoid multiplications having too small results. This is accomplished as in Figure 2c, resorting to a *Leading Zeros Recognizer* (LZR) based on a carry-lookahead Leading Zero Counting structure proposed in [12].

3.4 Hybrid Equal Weights and Approximation Skipping

The last proposed architecture is a combination of the Equal Weights Skipping and the Approximation Skipping techniques and it is depicted in Figure 2d. Thanks to this configuration, even more multiplications can be skipped.

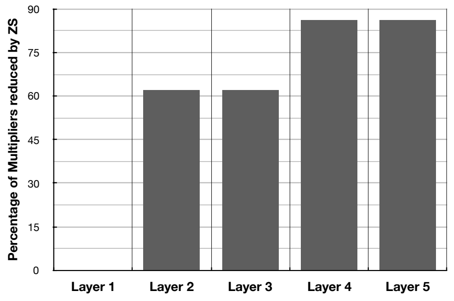
4 Experimental results

The following section presents the results of simulation, validation and synthesis. Indeed, the architecture has been simulated in order to verify the correct behavior, then validated on the well known AlexNet NN [3]. Finally, the architecture has been synthesized and simulated, thus the occupied area and power consumption, which will be discussed later.

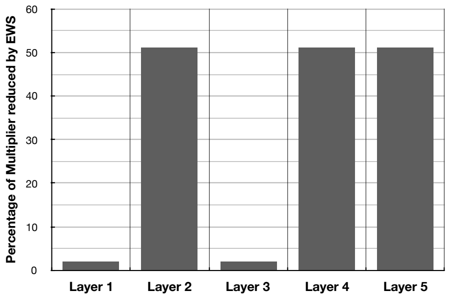
4.1 Validation

The proposed architecture has been validated on AlexNet [3], with the aim of testing the performance of the developed techniques on an existing test case. Such NN is composed of a total of 8 layers, but only the first five are convolutional, thus they represent where the proposed techniques are applied. The mentioned NN, already trained on ImageNet [13], has been described both in

Matlab and VHDL. In such way, every time a new technique is applied to the VHDL code, the performance is evaluated by comparing the results with the exact model in Matlab.
The first analysis is performed on the Zero Skipping Architecture. As it is possible to notice from Figure 3a, multiplications are significantly reduced. Since in the first layer no zero-padding is applied, no reduction is noticed, but in the successive layers, both for zero-padding and ReLU, the amount of skipped operations increases until the 85%.
The next approach to be analyzed is the Equal Weights Skipping. Such technique is not as efficient as the previous one (see Figure 3b), in fact, this method is strictly coupled with kernels and so with the NN model. In this case, layers 2, 4 and 5 allow for about 50% reduction of multiplications.



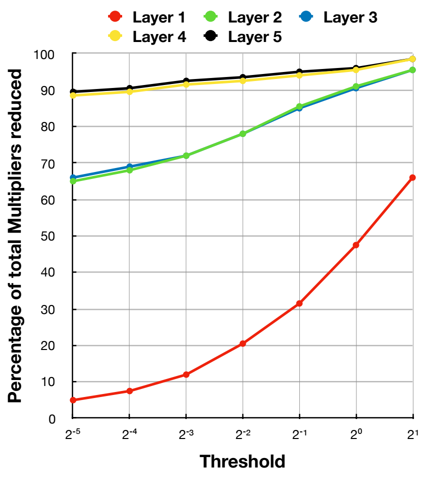
(a) ZS



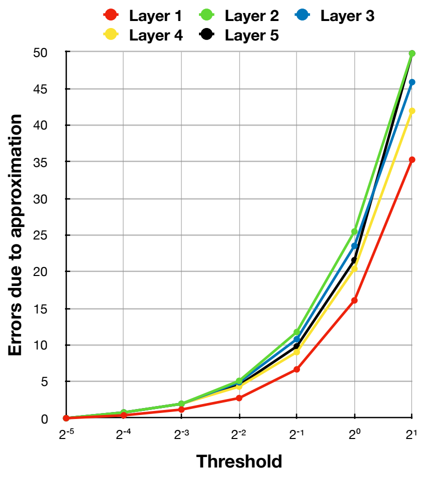
(b) EWS

Fig. 3: Multiplications reduction across AlexNet layers.

For what concerns the approximated architectures, a study on the range of approximation has been conducted with several thresholds going from [-2^{-5}:2^{-5}] to $[-2 : 2]$. The idea is to measure the impact of the approximation on the accuracy of the network and to evaluate the complexity reduction, i.e. the number of multiplications, not only with various threshold values but also with uniform and non-uniform thresholds for different layers.
Considering uniform boundaries, simulations showed that the NN is able to produce acceptable results with an accuracy around the 90% until [-2^0:2^0]. Since the sparsity of the features maps increases going deeper into the NN, the amount of saved multiplications increases as well in the last layers. In Figure 4a it is possible to notice the behavior of the Approximation Skipping architecture respect to the thresholds in all the five different layers. Obviously, the error (figure 4b) intensifies moving the threshold towards the integer part of the binary representation. Such an error is calculated as the normalized average of the absolute difference between the pixel values of the ideal and the approximate feature map.



(a)



(b)

Fig. 4: Left: multiplication reduction across AlexNet layers due to Approximation Skipping Architecture. Right: normalized average error

The Hybrid solution (Equal Weights and Approximation Skipping Architecture) applied to AlexNet [3], does not provide any clear improvements with respect to the previous architectures.
Regarding the non-uniform thresholds, no significant improvements have been noticed. This is mainly due to the simplicity of the NN and the fact that the layers are not independent as previous approximations affect next layers. So, in the first layers small-magnitude thresholds are needed, as such layers are trying to extract as many features as possible. So the last layers could have higher thresholds, but this approach is not very useful because in such layers the sparsity is very high and a Zero Skipping architecture is already enough.

4.2 Synthesis

The synthesis of the architecture has been performed by using *Synopsys Design Compiler* with the UMC 65nm technology. After the logic synthesis, a simulation was run on the obtained netlist in order to verify the correct behavior.
As it is possible to notice from Table 1, the low-power approaches present a lower maximum frequency compared to the starting architecture and an increased area. The size of the area increases going towards more complex solutions such as the hybrid one.
Here, a simulation is run again exploiting *Moldelsim*, recording the switching activity, which is fundamental to obtain an accurate estimation of the power consumption. In the following Table 2, a comparison between developed architectures and other works is presented:

Architecture	T_{clock}[ns]	F_{max}[MHz]	Area[μm^2]
Starting	1.45	689.65	40948.20
Zero Skipping	1.53	653.59	41575.76
Equal Weights Skipping	1.53	653.59	42182.64
Approximation Skipping	1.53	653.59	47296.08
Hybrid EW and APP Skipping	1.53	653.59	51039.68

Table 1: Frequency and area of the proposed achitectures.

Accelerators	Technology node [nm]	Bid Width	CLK Frequency [MHz]	Power [mW]
Starting ([14])	65	16	500	59
Starting (**This work**)	65	16	689.65	**21**
Zero Skipping ([14])	65	16	500	38
Zero Skipping (**This work**)	65	16	653.39	**20**
Approximation Skipping ([14])	65	16	500	31
Approximation Skipping (**This work**)	65	16	653.59	**19**
CPU Core-i7 5930k [15]	22	not given	3500	73000
GPU GeFore Titan X [15]	28	not given	1075	159000
mGPU Tegra K1 [15]	28	not given	852	5100

Table 2: Comparison with existing platforms

From table 2 is clear how the ASIC solutions lead to a reduced power consumption with respect to CPUs and GPUs, even though the frequency is about the half. Presented architectures perform slightly better than those in [14], even though the number of multipliers is 9 and 256, respectively.

5 Conclusion

This work proposes several hardware architectures for convolutional neural networks able to address the problem of the matrix sparsity caused by non-linear activation function like ReLU. Such architectures are capable of avoiding unnecessary operations like zero multiplications. Moreover, this paper presents an approximate technique based on leading-zeros able to skip operations involving parameters lower than a certain preset threshold.
After a detailed introduction of the different approaches, a validation, conducted on a well know network like AlexNet, is examined. Performance are compared doing a deep analysis of the NN layers, ranging through several thresholds. Synthesis results present similar performance with respect to other existing architecture, reaching even better low-power levels.
The percentage of skipped multiplications is encouraging in all the simulations, reaching up to the 85%. Such results demonstrate that the matrix sparsity in NNs could be exploited to further reduce the power consumption thus enabling new low-power frontiers.
However, the proposed architectures could be still improved by applying such techniques to a scheduler able to skip multiplication completely, reducing the latency besides the power. Anytime an operation must be skipped, a different one

is scheduled at its place increasing the execution speed. Such scheduler would require major modification of the initial structure.

References

1. O. Rajabi Shishvan, D. Zois, and T. Soyata. Machine intelligence in healthcare and medical cyber physical systems: A survey. *IEEE Access*, 6:46419–46494, 2018.
2. Y. Le Cun, L. D. Jackel, B. Boser, J. S. Denker, H. P. Graf, I. Guyon, D. Henderson, R. E. Howard, and W. Hubbard. Handwritten digit recognition: applications of neural network chips and automatic learning. *IEEE Communications Magazine*, 27(11):41–46, Nov 1989.
3. Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Neural Information Processing Systems*, 25, 01 2012.
4. Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv 1409.1556*, 09 2014.
5. C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, June 2015.
6. K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, June 2016.
7. Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. The MIT Press, 2016.
8. V. Sze, Y. Chen, T. Yang, and J. S. Emer. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329, Dec 2017.
9. J. Jo, S. Kim, and I. Park. Energy-efficient convolution architecture based on rescheduled dataflow. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 65(12):4196–4207, Dec 2018.
10. Y. 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*, 52(1):127–138, Jan 2017.
11. D. Kim, J. Ahn, and S. Yoo. Zena: Zero-aware neural network accelerator. *IEEE Design Test*, 35(1):39–46, Feb 2018.
12. G. Dimitrakopoulos, K. Galanopoulos, C. Mavrokefalidis, and D. Nikolos. Low-power leading-zero counting and anticipation logic for high-speed floating point units. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 16(7):837–850, July 2008.
13. J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.
14. Yuxiang Huan, Yifan Qin, Yantian You, Lirong Zheng, and Zhuo Zou. A low-power accelerator for deep neural networks with enlarged near-zero sparsity. 05 2017.
15. Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A. Horowitz, and William J. Dally. EIE: Efficient Inference Engine on Compressed Deep Neural Network. *arXiv e-prints*, page arXiv:1602.01528, Feb 2016.