Approximated Prediction Strategy for Reducing Power Consumption of Convolutional Neural Network Processor

Takayuki Ujiie Masayuki Hiromoto Takashi Sato
Graduate School of Informatics, Kyoto University
Yoshida-hon-machi, Sakyo, Kyoto, Japan
paper@easter.kuee.kyoto-u.ac.jp

Abstract

Convolutional neural network (CNN) is becoming popular because of its great ability for accurate image recognition. However, the computational cost is extremely high, which increases power consumption of embedded CV systems. This paper proposes an efficient computing method, LazyConvPool (LCP), and its hardware architecture to reduce power consumption of CNN-based image recognition. The LCP exploits redundancy of operations in CNN and only executes essential convolutions by an approximated prediction technique. We also propose Sign Connect, which is a low computational-cost approximated prediction without any multiplications. The experimental evaluation using image classification dataset shows that the proposed method reduces the power consumption by 17.8%–20.2% and energy consumption by 11.4%–14.1% while retaining recognition performance.

1. Introduction

Image recognition is a key technology in various fields such as robotics, automotive, security, and medical applications. Realizing efficient embedded systems for such applications is nontrivial because we must co-design both algorithm and hardware architecture. A large number of dedicated vision processors have been proposed, which utilize handcrafted features (Haar-like features [22], SIFT [20], HOG [8], etc.) and learning algorithms (SVM [6], Boosting [9], etc.). However, the design of the specialized processor is costly in terms of development efforts albeit it could potentially achieve good performance.

In contrast to the traditional recognition algorithms that utilize the handcrafted features, representation learning [1] is becoming increasingly popular in the state-of-the-art recognition methods. The key idea of the representation learning is to let the system learn the feature by itself, in addition to learning the classifier. Convolutional neural network (CNN) is one of those representation learning methods and has achieved remarkable results in various image recognition tasks [10, 15, 17]. CNN computes 2D convolutions on images over and over again to extract the features. The local features of images are acquired by adjusting the weights of the convolution filters. This general framework makes it possible for CNN to accurately recognize various classes of images without giving the manually designed features. In order to achieve even higher accuracy, the network structure of the CNN tends to be made deeper and deeper, requiring huge computational resources. An efficient computing scheme to facilitate such a large number of convolutions is desired to realize accurate image recognition on energy-limited embedded systems.

Among the several kinds of approaches for realizing energy-efficient CNN, recent researches focus on the inherent resilience of CNN [21, 23]. Calculation of pattern recognition algorithms generally contains redundant calculation to ensure the robustness against noisy inputs. In other words, the redundancy in most of the recognition algorithms can be pursued to reduce computational resources. In this work, we focus on the characteristic of the max pooling operation, which is an integral component of CNN that selects a representative feature in a small region. Specifically, we propose LazyConvPool (LCP), a new computation method for CNN's convolution layers and pooling layers. In LCP, only the essential convolutions that affect the final result are executed on the basis of the prediction results by approximated computations. The key features of the proposed LCP are as follows:

- LCP predicts a local region that will be selected later in the pooing layer to limit the application of the accurate and thus expensive convolutional operations to the predicted region.
- The prediction is based on the newly proposed *Sign Connect*, which gives a significantly lightweight approximation of the convolution with no multiplication operation, while keeping prediction accuracy.

- An efficient hardware architecture is also proposed to implement a low power LCP-based CNN with a minimal overhead.

The rest of the paper is organized as follows. Section 2 provides an overview of the related work. Section 3 describes the proposed methods, LCP and Sign Connect. Section 4 presents the hardware architecture for the proposed method and shows an implementation details to improve performance. Section 5 shows the experimental results, and finally, Section 6 concludes the paper.

2. Related works

2.1. Convolutional neural network

A typical CNN consists of convolution layers, pooling layers, and a fully connected layer as shown in Figure 1. First, in the convolution layers, 2D convolution is computed as

$$W * x(m,n) = \sum_u \sum_v W(u,v) x(m+u, n+v), \quad (1)$$

where x is an input image and W is a weight matrix of the convolution filter. The operator '$*$' means 2D convolution. This layer extracts local features of images to generate feature maps by using a large number of different filters. Then in the pooling layers, a representative feature value is selected from a small region called a pooling window. The pooling layer is useful for translation invariance and reduction of computational costs. As the representative value, maximum (max) or average values are commonly used. In this paper, we use max pooling, which selects the max value from the pooling window. After the convolution and the pooling, non-linearity is used as an activation function similarly to the traditional neural networks. Here, piecewise continuous functions are generally used as the activation function. In practice, rectified linear unit (ReLU) is widely used because it does not saturate gradients while requiring a small amount of computation. ReLU is represented as

$$f(x) = \max(0, x). \quad (2)$$

In the fully connected layer, an output $\boldsymbol{x}^{(m)}$ is computed from the outputs of the previous layer $\boldsymbol{x}^{(m-1)}$ as

$$\boldsymbol{x}^{(m)} = f(W^{\mathrm{T}} \boldsymbol{x}^{(m-1)} + \boldsymbol{b}), \quad (3)$$

where W is a weight matrix, $\boldsymbol{b}$ is a bias vector in this layer, and $f(\cdot)$ is an activation function. At the final stage of the network, the softmax function is often used as the activation function for multi-class classification:

$$f(x_i) = \frac{\exp(x_i)}{\sum_j^{n-1} \exp(x_j)} \quad (i = 0, \cdots, n-1). \quad (4)$$

The softmax function has an advantage because it normalizes values in the final stage and facilitates error computation in the training phase. The number of the output units in the final stage is equal to the number of the labels, and the classification result y is given by

$$y = \arg\max_i x_i. \quad (5)$$

2.2. Computationally efficient neural networks

CNN requires large computational costs as well as other neural networks. It is because a huge number of multiplications are executed to propagate signals in the networks. Some recent works are trying to tackle this problem by replacing the multiplications with simpler operations. Network pruning [14] and network compression [2, 3] reduce the amount of weight parameters by compressing networks. These methods try to compress networks by utilizing some representative values to describe the whole weight matrix. On the other hand, BinaryNet [7] and Ternary Connect [19] are the methods that limit the weight values to $\{-1, +1\}$ or $\{-1, 0, +1\}$ during the training phase in order to reduce the memory usage and computation complexity. In the following paragraph, we describe the more details of Ternary Connect, on which our proposed method Sign Connect is based.

Ternary Connect Ternary Connect is a computation method for neural networks that eliminates multiplications in the propagation. Ternary Connect replaces the original real weight $w(u,v) \in [-1,1]$ by an alternative weights $W(u,v) \in \{-1,0,1\}$, which is stochastically determined by $w(u,v)$. If $w(u,v) > 0$, the probability of the $W(u,v)$ value is given by

$$\begin{cases} P(W(u,v)=1) & = & w(u,v) \\ P(W(u,v)=0) & = & 1-w(u,v) \end{cases}. \quad (6)$$

Otherwise, if $w(u,v) < 0$, the probability is given by

$$\begin{cases} P(W(u,v)=-1) & = & -w(u,v) \\ P(W(u,v)=0) & = & 1+w(u,v) \end{cases}. \quad (7)$$

By using $W(u,v)$ in propagation computation in stead of $w(u,v)$, multiplications are completely omitted.

Although Ternary Connect is very useful to reduce computation in the training phase, it can deteriorate recognition performance if it is simply applied to the classification phase. In addition, a suitable hardware architecture should be considered at the same time in order to receive the benefit of reducing multiplications.

2.3. CNN processors

CNN in embedded systems has been actively researched in several aspects. As application-specific processors, some

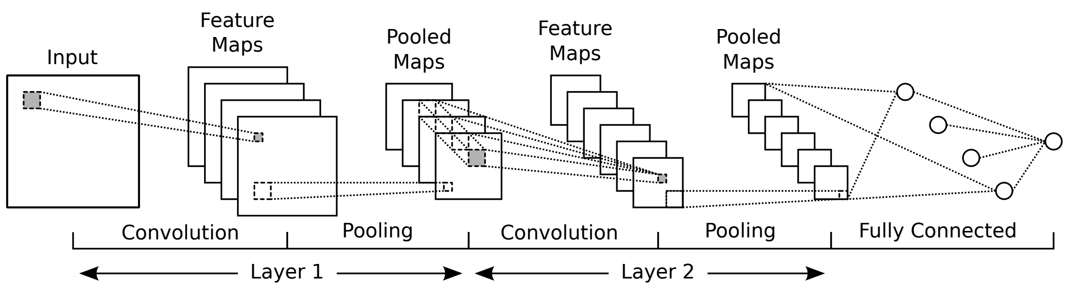

Figure 1: A typical architecture of a convolutional neural network. It consists of multiple sets of convolution and pooling layers and succeeding fully connected multi-layer perceptrons.

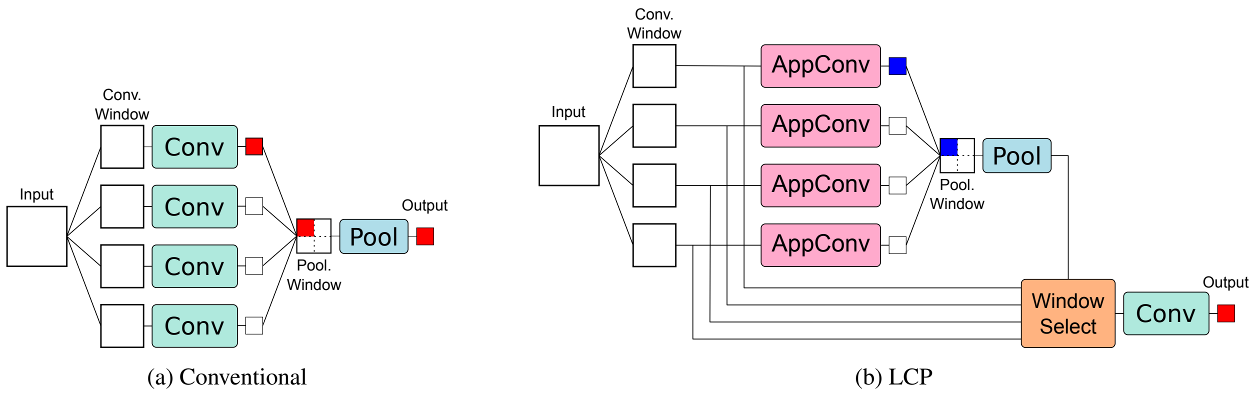

Figure 2: Convolution and pooling operations of the conventional method and the proposed method, LazyConvPool (LCP). (a) The conventional one executes exact convolutions (Conv) against all the windows. (b) LCP only executes an exact convolution (Conv) against the specified window predicted by the approximate convolution (AppConv).

architectures that implement CNN are proposed in [4, 5, 11, 13]. nn-X [11] is a salable co-processor for CNN. It consists of an array of possessing elements, which are dedicated hardware to accelerate the main components of CNN (convolution, pooling, and activation). HWCE [5] is also an accelerator for CNN particularly focusing on 2D convolutions, which are the most computationally intensive part in CNN. Both of the architectures intend to speed up the calculation of convolutions in a highly parallel manner by adopting an array structure and pipelined streaming processing units.

Another approach for low-power CNN processors is based on *approximate computing*, which is a computing method that aggressively uses approximation for the computation that does not require exact results. This is useful for the applications such as image processing and recognition tasks that tolerate some errors in the output. ApproxANN [23] and AxNN [21] are examples of using the approximate computing with neural networks. However, it is difficult to simply apply the approximate computing to CNN without spoiling recognition accuracy because of the deep structure of the CNN, where the approximation errors are accumulated and inflated during the propagation.

In this work, we propose a novel method to enhance the energy efficiency of the existing CNN processor like nn-X and HWCE by utilizing an approximation technique. In order to suppress the degradation of recognition performance, we adopt the approximation only to predict the region to be selected in the max pooling. With this approach, we can effectively eliminate redundant computation according to the prediction results calculated with a small overhead.

3. Proposed method

In this section, we propose LazyConvPool (LCP), an approximate computational method that combines convolution and max pooling for reducing power consumption of CNN processors. We also propose Sign Connect, a simplified arithmetic unit that can be effectively used with LCP.

3.1. LazyConvPool

The objective of LCP is to eliminate the convolutional computation that will not affect the output of the succeeding pooling layer. Figure 2a illustrates the conventional convolution and pooling operations. In this example, four convolutions are first executed on an input image to calculate adjacent four feature values in a 2×2 pooling window. Then the max pooling is operated to pick up one representative feature that has a maximum value (shown as a red pixel). Worthy of attention here is that only one convolution result is passed to the next stage and all the other convolution results are discarded by the pooling operation. This means that the non-representative feature values do not affect the feature map after pooling as long as the feature selection in the pooling is correct.

On the basis of this observation, we construct LCP framework with two main parts: (1) approximated convolution units (AppConv) to predict the feature selection in the pooling layer, and (2) a single regular (and exact) convolution unit (Conv) to perform convolution only on the predicted window. Figure 2b illustrates the LCP-based convolution and pooling operations. The computation flow of LCP is summarized as follows:

Step 1 Compute AppConv using the lightweight and approximate computation.

Step 2 Operate max pooling to the results of AppConv to specify the convolution index of which the convolution result is propagated.

Step 3 Compute Conv only to the input corresponding to the specified index in Step 2.

LCP can reduce the number of the exact convolutions by $1/N_{\mathrm{P}}$, where N_{P} is the number of the features in a pooling window. On the other hand, the prediction by AppConv introduces additional hardware. To minimize additional power consumption associated with the prediction, we utilize lightweight and approximate computation called Sign Connect, which is described in the next subsection.

3.2. Sign Connect

LCP executed only the necessary convolution by computing AppConv before Conv. Computational cost of AppConv must be low to realize power reduction through LCP. Here, for this purpose, we propose Sign Connect as an approximate computation unit of reduced computation complexity.

Sign Connect is equivalent to "sign function" used in BinaryNet [7]. Sign Connect computes product terms by the following equations:

$$\begin{cases} P(W(k,l)=1) & = & 1 \\ P(W(k,l)=0) & = & 0 \end{cases}, \tag{8}$$

$$\begin{cases} P(W(k,l)=-1) & = & 1 \\ P(W(k,l)=0) & = & 0 \end{cases}. \tag{9}$$

Contrary to Ternary Connect [19], the proposed equations are deterministic, thus random number generators are not necessary. Sign Connect can be implemented so that general multipliers are replaced by the multiplications of the sign of weights, which can be typically realized by multiplexers. With Sign Connect, multiplication could be drastically simplified, and the total performance becomes deterministic.

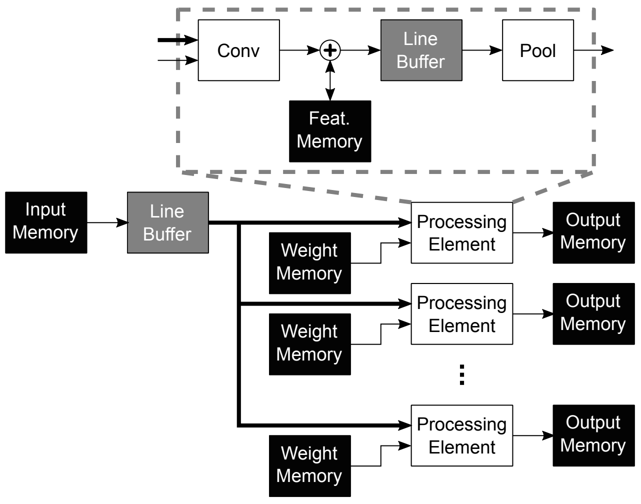

Figure 3: Base architecture for CNN processor.

4. Hardware architecture

4.1. Base architecture

Figure 3 shows a block diagram of the base architecture of CNN processor. This is similar to those of nn-X [11] and HWCE [5], employing a parallel architecture with a shared memory and multiple processing elements. Each processing element, connected to the shared memory, executes convolution, pooling, and activation. In this hardware architecture, each element computes individual feature maps on the basis of a common input image (or feature map). The input image is supplied to each element via a line buffer. The line buffer enables a continuous and efficient data feed for convolutions on a sliding windows. Weights of the convolution layers are supplied from a local memory and kept in registers in convolution circuits during computation. At the following stage of convolution circuits, the memory and accumulators are connected to accumulate the value of feature maps. Convolution results are accumulated on the memory that has the size of the feature map.

4.2. LCP architecture

Figure 4 shows the proposed processing element architecture in which LCP is applied to the base architecture. Buffered inputs are supplied to AppConv and Conv with weights used for convolution. A processing element first computes AppConv and pooling. The results are temporarily stored in the index memory connected to the pooling circuit and utilized as the signal that control inputs to the Conv

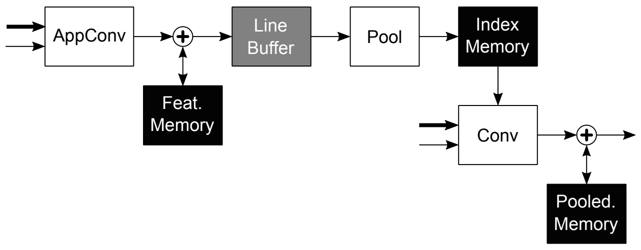

Figure 4: The proposed architecture for LCP-based CNN processor.

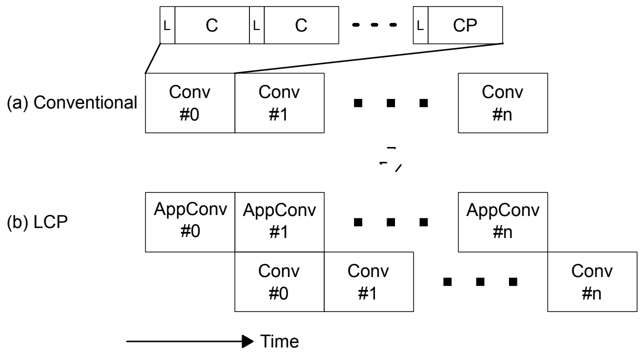

Figure 5: Timing chart of CNN calculation by (a) the conventional and (b) the proposed LCP methods.

circuit. Conv results are accumulated only for the size of the pooled feature map.

Figure 5 shows the timing charts of CNN computation with and without LCP. As shown in Figure 5a, the conventional CNN on the base architecture repeatedly executes sets of convolutions. Each set consists of weight loading (L) and convolution (C). In the final set, pooling is executed just after the convolutions (CP). As for LCP, the two kinds of convolutions (AppConv and Conv) must be processed and this may extend the execution time. In order to minimize the additional execution time caused by LCP, the proposed architecture conducts the two kinds of convolutions in a pipelined way as shown in Figure 5b. Thus, the overhead of execution time is reduced to just one computing routine time.

5. Experiments

5.1. Classification accuracy

Datasets We evaluated the classification accuracy of LCP with image recognition datasets. We used two datasets: the handwritten digits dataset MNIST [18] and the object recognition dataset CIFAR-10 [16]. For MNIST, we preprocessed the images by the normalization to limit the range of data values to 0–1. For CIFAR-10, we preprocessed images by a) global contrast normalization, b) ZCA whitening [16], and c) horizontal flips. a) and b) are preprocessing techniques for normalization. c) is applied for data augmentation without changing the image size. We trained networks by all data given in the training set and evaluated target methods by 100 data in the test set none of which exists in the training set. Network parameters are shown in Tables 1 and 2.

Layer name	Parameter
Input image	size: 28x28, channel: 1
Convolution	kernel: 5x5, channel: 20
Max pooling	kernel: 2x2, stride: 2
ReLU	
Convolution	kernel: 5x5, channel: 50
Max pooling	kernel: 2x2, stride: 2
ReLU	
Fully connected	channel: 500
ReLU	
Fully connected	channel: 10
Softmax	

Table 1: Network parameters of CNN for MNIST dataset.

Layer name	Parameter
Input image	size: 32x32, channel: 3
Convolution	kernel: 5x5, channel: 64, padding: 2
Max pooling	kernel: 2x2, stride: 2
ReLU	
Convolution	kernel: 5x5, channel: 128, padding: 2
Max pooling	kernel: 2x2, stride: 2
ReLU	
Convolution	kernel: 5x5, channel: 256, padding: 2
Max pooling	kernel: 2x2, stride: 2
ReLU	
Convolution	kernel: 5x5, channel: 512, padding: 2
Max pooling	kernel: 2x2, stride: 2
ReLU	
Convolution	kernel: 5x5, channel: 512, padding: 2
Max pooling	kernel: 2x2, stride: 2
ReLU	
Fully connected	channel: 512
ReLU	
Fully connected	channel: 10
Softmax	

Table 2: Network parameters of CNN for CIFAR-10 dataset.

Results Classification results are shown in Table 3. We evaluated the four types of CNN: (a) the conventional CNN, (b) CNN with LCP that computes AppConv by the exact convolution, (c) AppConv by Ternary Connect, and (d) AppConv by Sign Connect. The results of LCP with Ternary

Method	MNIST	CIFAR-10
(a) Conventional	0.986	0.860
(b) LCP	0.986	0.860
(c) LCP + TC	0.984	0.825
(d) LCP + SC	0.988	0.849

Table 3: Classification accuracy on datasets.

Connect are the average of 10 independent runs because of the probabilistic nature of the Ternary Connect.

The results for LCP with exact convolution is completely equal to the conventional one because this is equivalent to the addition of redundant operation. For LCP with Ternary Connect, the accuracy is almost the same in the experiment of MNIST. However, the accuracy degrades about 3.5% in CIFAR-10. The difference shows the performance degradation due to the prediction inaccuracy using Ternary Connect for the complex data such as color images. For LCP with Sign Connect, the accuracy is almost equal to the conventional result in MNIST data set. Regarding to CIFAR-10, the accuracy degradation has been reduced compared to LCP with Ternary Connect.

5.2. Power consumption

Experimental setup To evaluate power consumption, we implemented both of the base architecture and the proposed LCP architecture with hardware description language (HDL). The dynamic power consumption is evaluated by using the netlists synthesized from the HDL and the logic simulation results that annotate the switching activity in the netlists. We used two commercial logic cell libraries of 65 nm process at 0.55 V and 28 nm process at 0.92 V.

In the power estimation, we focused on a mid-level layer when recognizing 10 MNIST samples. The numbers of input and output feature maps for this layer are 20 and 50, respectively. CNN processor implements $P = 4$ processing elements operating at a 100 MHz clock frequency.

Results Figures 6 and 7 show the average power consumption for each digit in the base and the proposed LCP architecture. We can see that the power consumption of the LCP architecture decreases for all digits.

Figures 8 and 9 show the average power consumption for all digits in the base and the proposed LCP architectures. The breakdowns of power consumption for major modules are also presented. It can be seen that the power consumption in Conv modules is greatly reduced by using LCP. Although the power consumption in AppConv and the other modules is increased, the total power decreases by 20.2% and 17.8% for 65 nm and 28 nm process libraries, respectively. Considering the increase of the execution time discussed in 4.2, which is 7.7% in this experiment, the energy reduction by LCP is 14.1% and 11.4% for 65 nm and 28 nm process libraries, respectively.

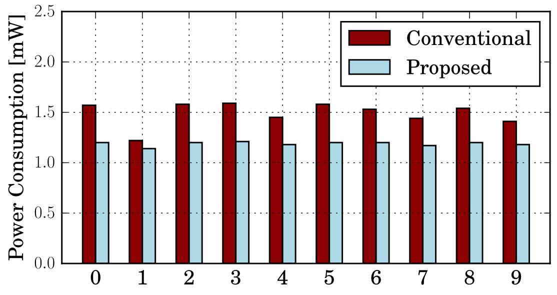

Figure 6: Power consumption for classifying each digit in the CNN architectures with 65 nm process library.

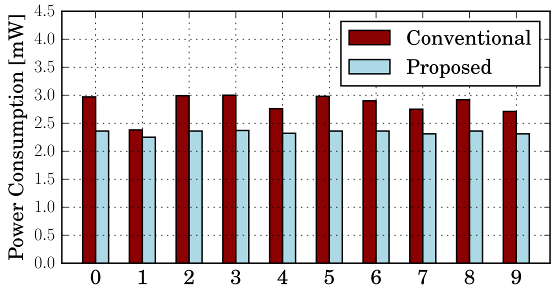

Figure 7: Power consumption for classifying each digit in the CNN architectures with 28 nm process library.

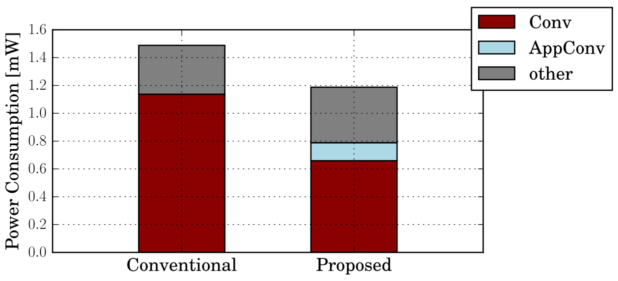

Figure 8: Average power consumption and breakdowns for the major modules in the CNN architectures with 65 nm process library.

6. Conclusion

In this paper, a new approach to reduce the number of multiplications in CNN based on an approximated prediction is proposed. Through the experimental evaluation, the proposed LazyConvPool algorithm and its hardware architecture successfully reduce the energy consumption for image recognition by 11.4%–14.1% while keeping the recognition accuracy. The reduction rate is limited in this pa-

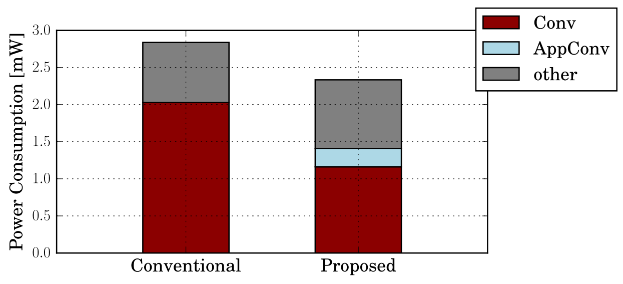

Figure 9: Average power consumption and breakdowns for the major modules in the CNN architectures with 28 nm process library.

per because we only focused on the pooling operations. For a future work, we will apply the proposed approach to other kinds of selective operations in neural networks such as Maxout techniques [12] and network compression techniques [3, 14] to realize more energy-efficient CNN processors.

References

[1] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, 2013. 1

[2] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen. Compressing convolutional neural networks. *Computing Research Repository*, abs/1506.04449, 2015. 2

[3] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen. Compressing neural networks with the hashing trick. In *Proceedings of International Conference on Machine Learning*, pages 2285–2294, 2015. 2, 7

[4] Y.-H. Chen, T. Krishna, J. Emer, and V. Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. In *Proc. IEEE International Solid-State Circuits Conference*, pages 262–263, 2016. 3

[5] F. Conti and L. Benini. A ultra-low-energy convolution engine for fast brain-inspired vision in multicore clusters. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition*, pages 683–688, 2015. 3, 4

[6] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995. 1

[7] M. Courbariaux and Y. Bengio. Binarynet: Training deep neural networks with weights and activations constrained to $+1$ or -1. *Computing Research Repository*, abs/1602.02830, 2016. 2, 4

[8] N. Dalai, B. Triggs, I. Rhone-Alps, and F. Montbonnot. Histograms of oriented gradients for human detection. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 886–893, 2005. 1

[9] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997. 1

[10] R. Girshick. Fast R-CNN. In *Proceedings of the IEEE International Conference on Computer Vision*, 2015. 1

[11] V. Gokhale, J. Jin, A. Dundar, B. Martini, and E. Culurciello. A 240 G-ops/s mobile coprocessor for deep neural networks. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshop*, pages 696–701, 2014. 3, 4

[12] I. Goodfellow, D. Warde-farley, M. Mirza, A. Courville, and Y. Bengio. Maxout networks. In *Proceedings of the International Conference on Machine Learning*, pages 1319–1327, 2013. 7

[13] 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. *Computing Research Repository*, abs/1602.01528, 2016. 3

[14] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *Computing Research Repository*, abs/1510.00149, 2015. 2, 7

[15] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *Computing Research Repository*, abs/1512.03385, 2015. 1

[16] A. Krizhevsky. Learning multiple layers of features from tiny images. Master's thesis, University of Toronto, 2009. 5

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In *Proceedings of the Neural Information Processing Systems*, pages 1097–1105. 2012. 1

[18] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 5

[19] Z. Lin, M. Courbariaux, R. Memisevic, and Y. Bengio. Neural networks with few multiplications. *Computing Research Repository*, abs/1510.03009, 2015. 2, 4

[20] D. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the IEEE International Conference on Computer Vision*, volume 2, pages 1150–1157, 1999. 1

[21] S. Venkataramani, A. Ranjan, K. Roy, and A. Raghunathan. AxNN: Energy-efficient neuromorphic systems using approximate computing. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 27–32, 2014. 1, 3

[22] P. Viola and M. Jones. Robust real-time object detection. *International Journal of Computer Vision*, 57(2):137–154, 2004. 1

[23] Q. Zhang, T. Wang, Y. Tian, F. Yuan, and Q. Xu. ApproxANN: An approximate computing framework for artificial neural network. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition*, pages 701–706, 2015. 1, 3