



DECLARATION OF GORDON MACPHERSON

I, Gordon MacPherson, am over twenty-one (21) years of age. I have never been convicted of a felony, and I am fully competent to make this declaration. I declare the following to be true to the best of my knowledge, information, and belief:

1. I am the Director, Board Governance & Policy Development at The Institute of Electrical and Electronics Engineers, Incorporated ("IEEE").
2. IEEE is a neutral third party in this dispute.
3. I am not being compensated for this declaration and IEEE is only being reimbursed for the cost of the articles I am certifying.
4. Among my responsibilities as Director of Board Governance & Policy Development, I act as a custodian of certain records for IEEE.
5. I make this declaration based on my personal knowledge and information contained in the business records of IEEE.
6. As part of its ordinary course of business, IEEE publishes and makes available technical articles and standards. These publications are made available for public download through the IEEE digital library, IEEE Xplore.
7. It is the regular practice of IEEE to publish articles and other writings including article abstracts and make them available to the public through IEEE Xplore. IEEE maintains copies of publications in the ordinary course of its regularly conducted activities.
8. The articles below have been attached as Exhibits A- Exhibit C to this declaration:

Exhibit A	T. Ujiie, M. Hiromoto and T. Sato, "Approximated Prediction Strategy for Reducing Power Consumption of Convolutional Neural Network Processor," <i>2016 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)</i> , Las Vegas, NV, USA, 2016, pp. 870-876, doi: 10.1109/CVPRW.2016.113.
Exhibit B	B. Moons, B. De Brabandere, L. Van Gool and M. Verhelst, "Energy-Efficient ConvNets Through Approximate Computing," <i>2016 IEEE Winter Conference on Applications of Computer Vision (WACV)</i> , Lake Placid, NY, USA, 2016, pp. 1-8, doi: 10.1109/WACV.2016.7477614.
Exhibit C	H. Kaul <i>et al.</i> , "14.4 A 21.5M-Query-Vectors/s 3.37nJ/Vector Reconfigurable K-Nearest-Neighbor Accelerator with Adaptive Precision in 14nm Tri-Gate CMOS," <i>2016 IEEE International Solid-State Circuits Conference (ISSCC)</i> , San Francisco, CA, USA, 2016, pp. 260-261, doi: 10.1109/ISSCC.2016.7418006.

9. I obtained a copy of Exhibit A through IEEE Xplore, where it is maintained in the ordinary course of IEEE's business. Exhibit A is a true and correct copy of the Exhibit, as it existed on or about September 19, 2025.
10. I obtained a copy of Exhibit B through IEEE Xplore, where it is maintained in the ordinary course of IEEE's business. Exhibit B is a true and correct copy of the Exhibit, as it existed on or about September 22, 2025.
11. I obtained a copy of Exhibit C through IEEE Xplore, where it is maintained in the ordinary course of IEEE's business. Exhibit C is a true and correct copy of the Exhibit, as it existed on or about September 22, 2025.
12. The articles and abstracts from IEEE Xplore show the dates of publication. IEEE Xplore populates this information using the metadata associated with the publication.
13. T. Ujii, M. Hiromoto and T. Sato, "Approximated Prediction Strategy for Reducing Power Consumption of Convolutional Neural Network Processor," was published in 2016 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), held on June 26, 2016 - July 1, 2016, in Las Vegas, NV, USA. Copies of the conference proceedings were made available no later than the last day of conference. This article was added to the IEEE digital library Xplore on December 19, 2016, as shown from the abstract from IEEE Xplore, and was publicly available for download from IEEE Xplore by December 19, 2016, and is currently available for public download from the IEEE digital library, IEEE Xplore.
14. B. Moons, B. De Brabandere, L. Van Gool and M. Verhelst, "Energy-Efficient ConvNets through Approximate Computing," was published in 2016 IEEE Winter Conference on Applications of Computer Vision (WACV), held on March 7-10, 2016, in Lake Placid, NY, USA. Copies of the conference proceedings were made available no later than the last day of conference. This article was added to the IEEE digital library Xplore on May 26, 2016, as shown from the abstract from IEEE Xplore, and was publicly available for download from IEEE Xplore by May 26, 2016, and is currently available for public download from the IEEE digital library, IEEE Xplore.
15. H. Kaul et al., "14.4 A 21.5M-Query-Vectors/s 3.37nJ/Vector Reconfigurable K-Nearest-Neighbor Accelerator with Adaptive Precision in 14nm Tri-Gate CMOS," was published in 2016 IEEE International Solid-State Circuits Conference (ISSCC), held on January 31, 2016 - February 4, 2016, in San Francisco, CA, USA. Copies of the conference proceedings were made available no later than the last day of conference. This article was added to the IEEE digital library Xplore on February 25, 2016, as shown from the abstract from IEEE Xplore, and was publicly available for download from IEEE Xplore by February 25, 2016, and is currently available for public download from the IEEE digital library, IEEE Xplore.

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true, and further that these statements were made with the knowledge that willful false statements and the like are punishable by fine or imprisonment, or both, under 18 U.S.C. § 1001.

I declare under penalty of perjury that the foregoing statements are true and correct.

Date: 9/22/2025

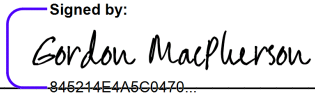
By:  Signed by:
845214E4A5C0470...

EXHIBIT A

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 net-

work (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 pooling 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 $\mathbf{x}^{(m)}$ is computed from the outputs of the previous layer $\mathbf{x}^{(m-1)}$ as

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

where W is a weight matrix, $\mathbf{b}$ 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:

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

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 instead 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

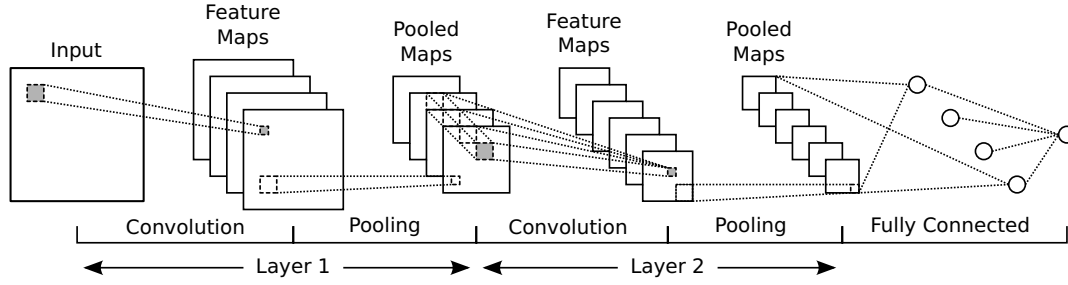


Figure 1: A typical architecture of a convolutional neural network. It consists of multiple sets of convolution and pooling layers and succeeding fully connected multi-layer perceptrons.

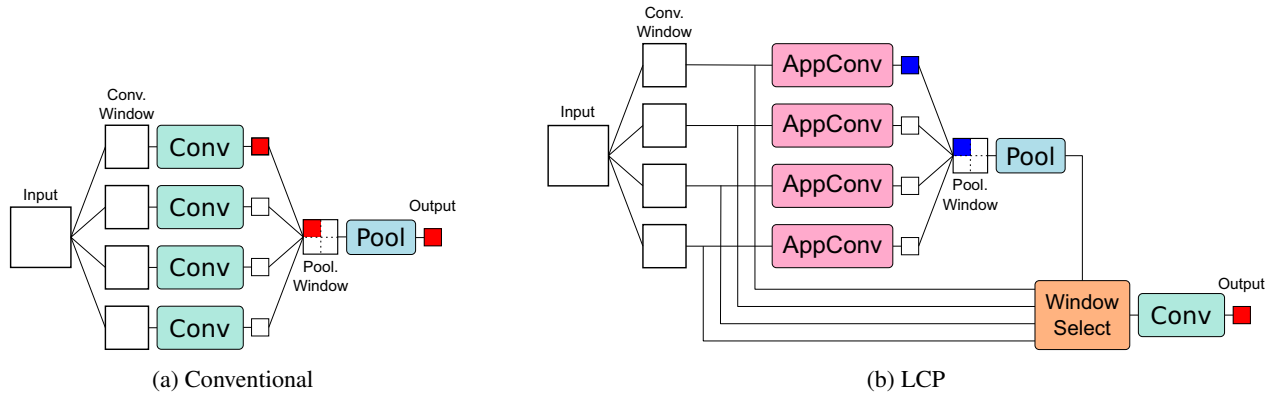


Figure 2: Convolution and pooling operations of the conventional method and the proposed method, LazyConvPool (LCP). (a) The conventional one executes exact convolutions (Conv) against all the windows. (b) LCP only executes an exact convolution (Conv) against the specified window predicted by the approximate 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 processing 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 $1/N_P$, 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:

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

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

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

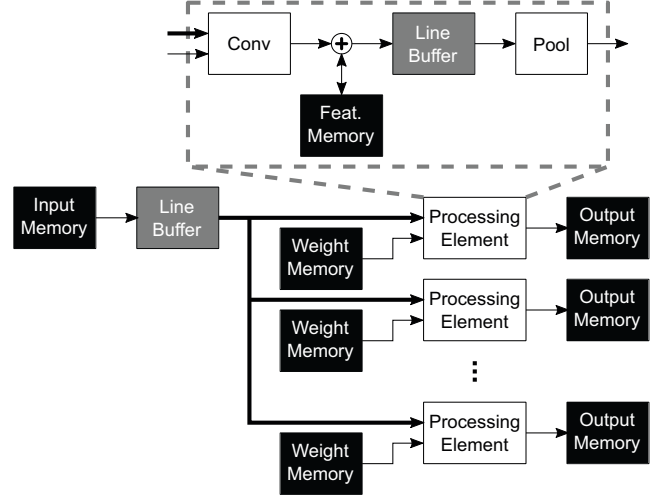


Figure 3: Base architecture for CNN processor.

sign of weights, which can be typically realized by multiplexers. With Sign Connect, multiplication could be drastically simplified, and the total performance becomes deterministic.

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

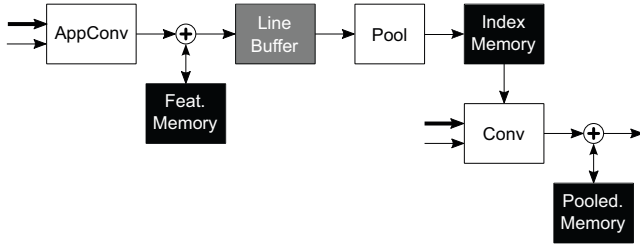


Figure 4: The proposed architecture for LCP-based CNN processor.

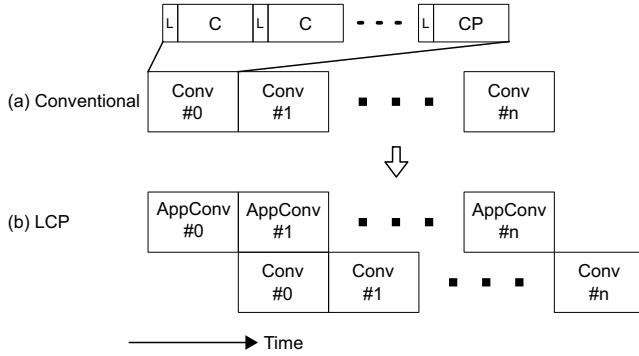


Figure 5: Timing chart of CNN calculation by (a) the conventional and (b) the proposed LCP methods.

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 pre-processed 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

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.

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.

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 en-

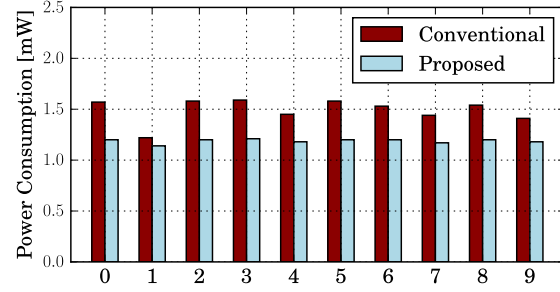


Figure 6: Power consumption for classifying each digit in the CNN architectures with 65 nm process library.

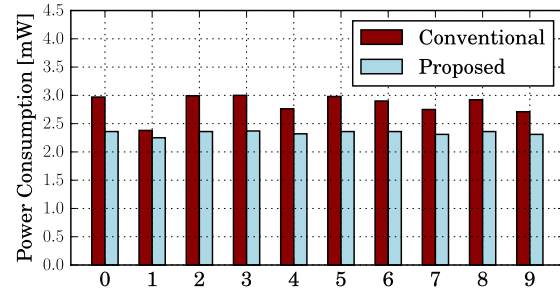


Figure 7: Power consumption for classifying each digit in the CNN architectures with 28 nm process library.

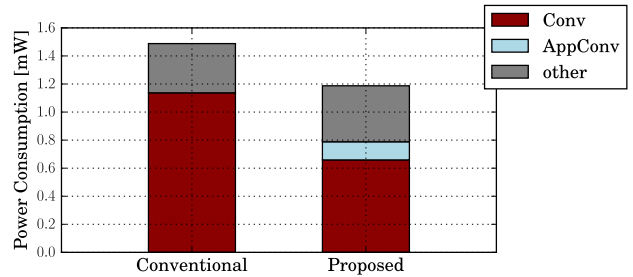


Figure 8: Average power consumption and breakdowns for the major modules in the CNN architectures with 65 nm process library.

ergy reduction by LCP is 14.1% and 11.4% for 65 nm and 28 nm process libraries, respectively.

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-

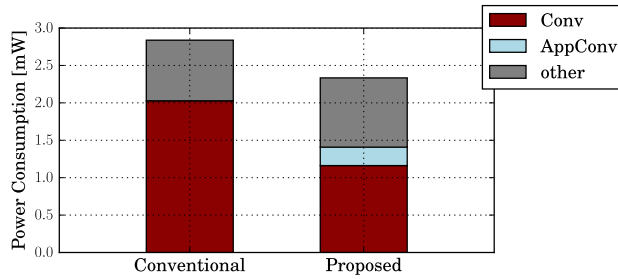


Figure 9: Average power consumption and breakdowns for the major modules in the CNN architectures with 28 nm process library.

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. Approx-ANN: 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

ABSTRACT A

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

Publisher: IEEE

[Cite This](#) PDFTakayuki Ujii ; Masayuki Hiromoto ; Takashi Sato [All Authors](#)10
Cites in
Papers419
Full
Text Views

Abstract

Document Sections

1. [Introduction](#)
2. [Related Works](#)
3. [Proposed Method](#)
4. [Hardware Architecture](#)
5. [Experiments](#)

[Show Full Outline](#) ▼[Authors](#)[Figures](#)[References](#)[Citations](#)[Keywords](#)[Metrics](#)[More Like This](#)

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.

Published in: [2016 IEEE Conference on Computer Vision and Pattern Recognition Workshops \(CVPRW\)](#)

Date of Conference: 26 June 2016 - 01 July 2016

DOI: [10.1109/CVPRW.2016.113](#)

Date Added to IEEE Xplore: 19 December 2016

Publisher: IEEE

► ISBN Information:

Conference Location: Las Vegas, NV, USA

Electronic ISSN: 2160-7516

SECTION 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 pooling 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.

SECTION 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

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

[View Source](#) ?

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

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

[View Source](#) ?

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

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

[View Source](#)

where W is a weight matrix, 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:

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

[View Source](#)

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

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

[View Source](#)

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

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

[View Source](#)

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

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

[View Source](#)

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 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 processing 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.

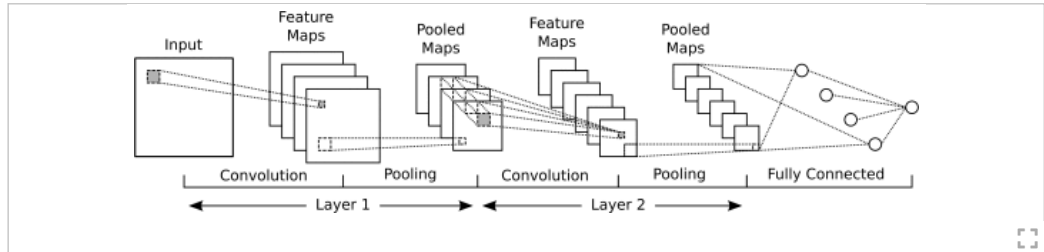


Figure 1:

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.

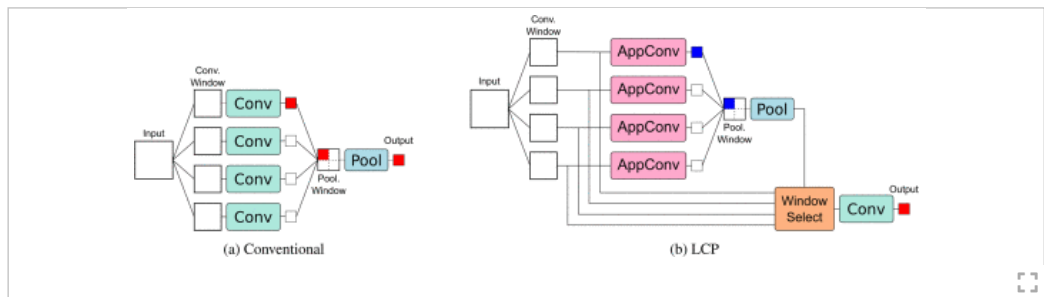


Figure 2:

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).

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. Ap-proxANN [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.

SECTION 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 $1/N_p$ where N_p is the number of the features in a pooling window. On the other hand, the prediction by App-Conv 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 App-Conv 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 Bi-naryNet [\[7\]](#). Sign Connect computes product terms by the following equations:

$$\begin{cases} P(W(k, l) = 1) = 1 \\ P(W(k, l) = 0) = 0 \end{cases} \tag{8}$$
$$\begin{cases} P(W(k, l) = -1) = 1 \\ P(W(k, l) = 0) = 0 \end{cases} \tag{9}$$

[View Source](#)

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.

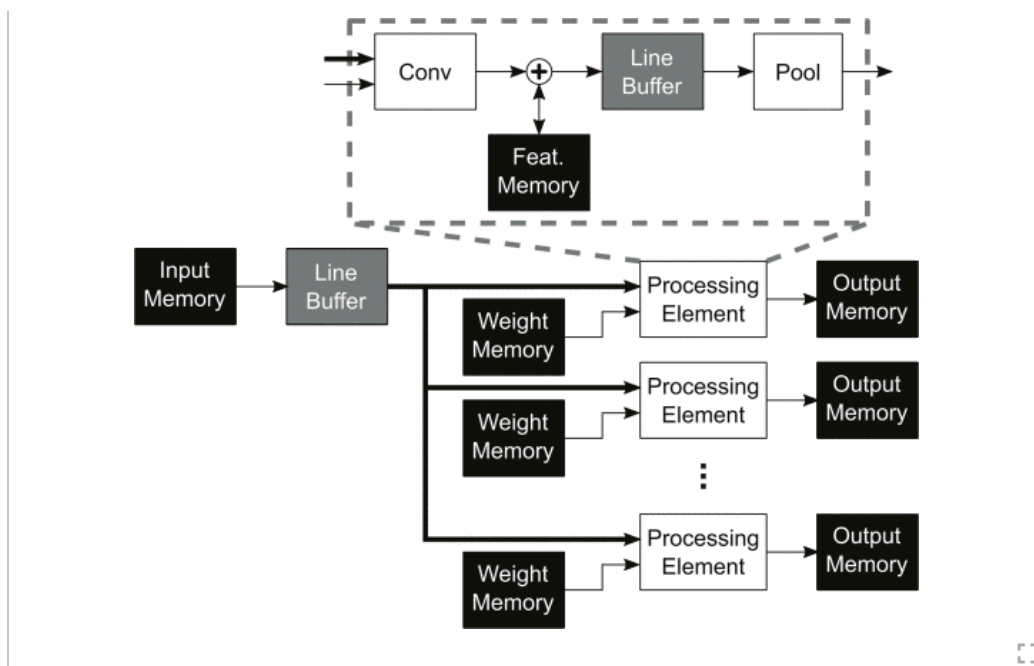


Figure 3:
Base architecture for CNN processor.

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

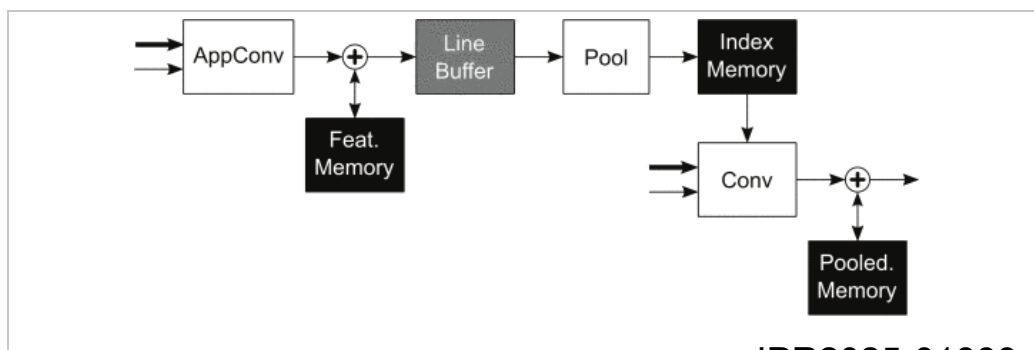


Figure 4:

The proposed architecture for LCP-based CNN processor.

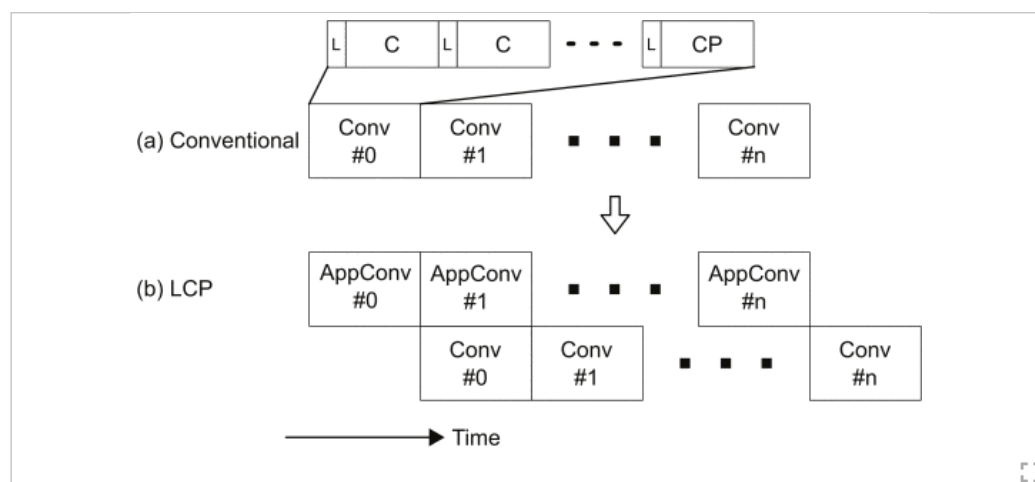


Figure 5:

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

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.

SECTION 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.

Table 1: Network parameters of CNN for MNIST dataset.

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 2: Network parameters of CNN for CIFAR-10 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	

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) Ap-pConv by Sign Connect. The results of LCP with Ternary Connect are the average of 10 independent runs because of the probabilistic nature of the Ternary Connect.

Table 3: Classification accuracy on datasets.

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

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.

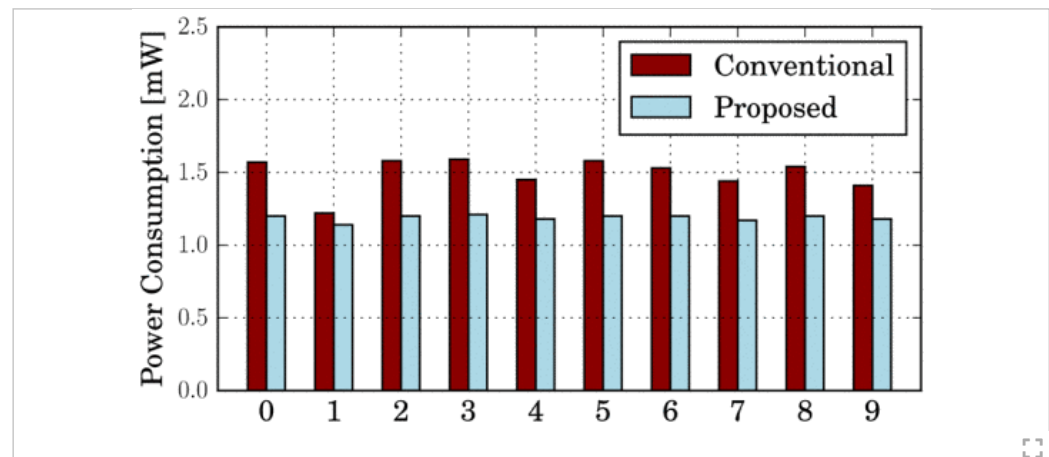


Figure 6:

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

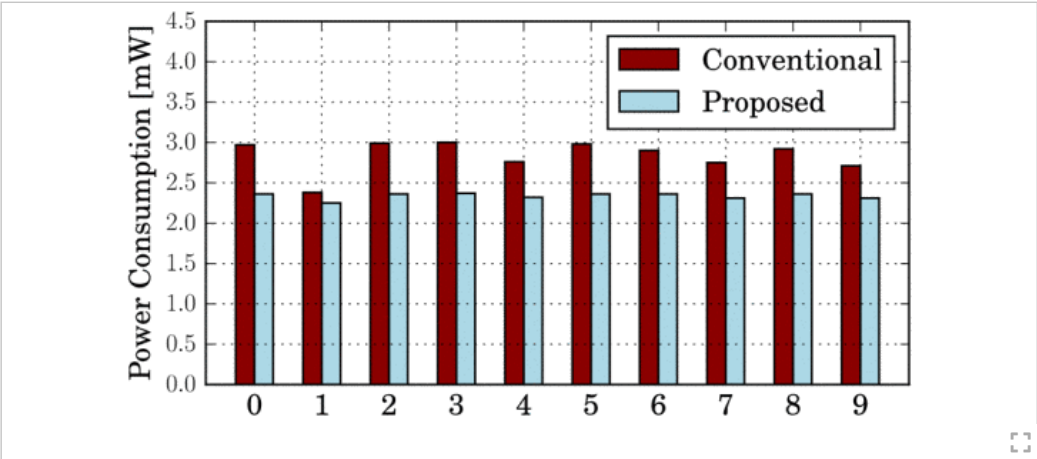


Figure 7: Power consumption for classifying each digit in the CNN architectures with 28 nm process library.

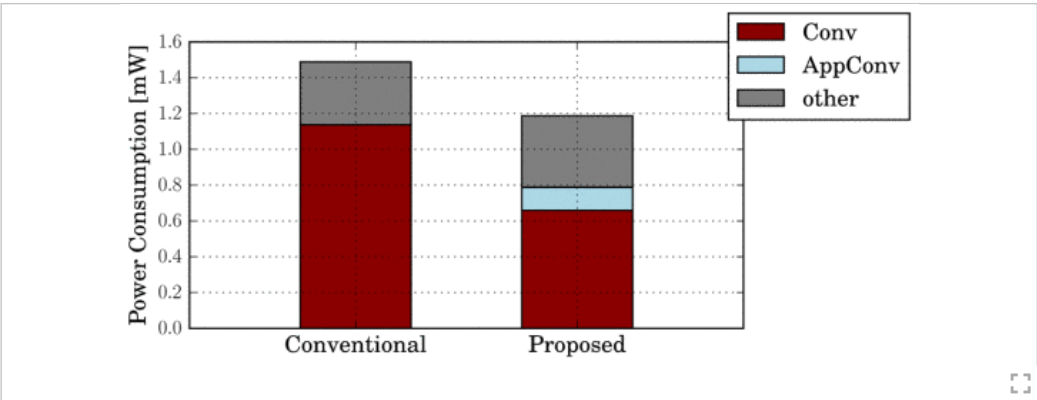


Figure 8: Average power consumption and breakdowns for the major modules in the CNN architectures with 65 nm process library.

SECTION 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 paper 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.

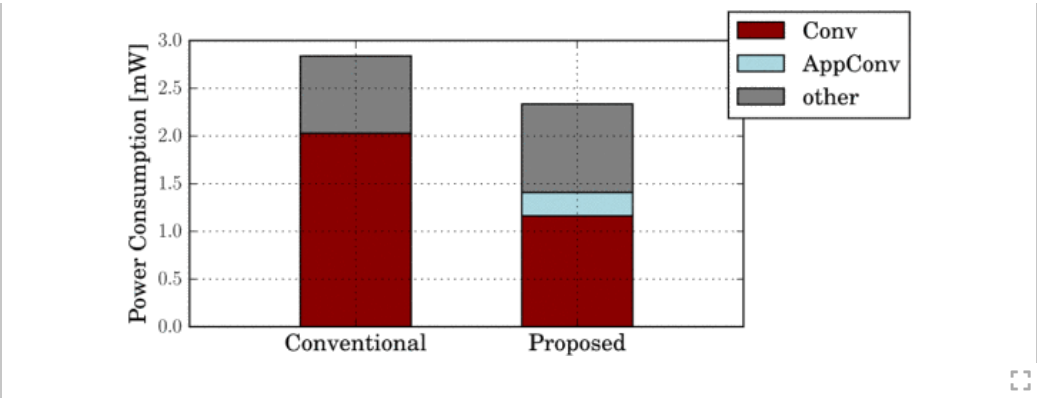


Figure 9:
Average power consumption and breakdowns for the major modules in the CNN architectures with 28 nm process library.

Authors	▼
Figures	▼
References	▼
Citations	▼
Keywords	▼
Metrics	▼

[Back to Results](#)

IEEE Personal Account	Purchase Details	Profile Information	Need Help?	Follow
CHANGE USERNAME/PASSWORD	PAYMENT OPTIONS VIEW PURCHASED DOCUMENTS	COMMUNICATIONS PREFERENCES PROFESSION AND EDUCATION TECHNICAL INTERESTS	US & CANADA: +1 800 678 4333 WORLDWIDE: +1 732 981 0060 CONTACT & SUPPORT	f @ in v

EXHIBIT B

Energy-Efficient ConvNets Through Approximate Computing

Bert Moons^{†*}

Bert De Brabandere^{°*}

Luc Van Gool[°]

Marian Verhelst[†]

[†]ESAT-MICAS, [°]ESAT-VISICS, Dept. of Electrical Engineering, KU Leuven

bert.moons@esat.kuleuven.be, bert.debrabandere@esat.kuleuven.be

Abstract

Recently convolutional neural networks (ConvNets) have come up as state-of-the-art classification and detection algorithms, achieving near-human performance in visual detection. However, ConvNet algorithms are typically very computation and memory intensive. In order to be able to embed ConvNet-based classification into wearable platforms and embedded systems such as smartphones or ubiquitous electronics for the internet-of-things, their energy consumption should be reduced drastically. This paper proposes methods based on approximate computing to reduce energy consumption in state-of-the-art ConvNet accelerators. By combining techniques both at the system- and circuit level, we can gain energy in the systems arithmetic: up to $30\times$ without losing classification accuracy and more than $100\times$ at 99% classification accuracy, compared to the commonly used 16-bit fixed point number format.

1. Introduction

Recently neural networks have made an impressive comeback in the field of machine learning. Convolutional neural networks or ConvNets are consistently pushing the state-of-the-art in areas like computer vision and speech processing. One of the reasons for this revival is the increasing availability of computing power. Multicore CPU's, GPU's, and even clusters of GPU's are no longer prohibitively expensive and make it possible to train and evaluate larger networks.

Unfortunately, the increase in computing power is inevitably accompanied by an increase in energy consumption. While high energy consumption is no big concern during the network's training phase - which typically takes place on a computer cluster - it poses a problem when the network needs to be evaluated on mobile hardware like smartphones, smart glasses and other wearable devices.

In this paper, we investigate how energy consumption can be reduced through principles of approximate computing.

Our analysis shows that it is possible to quantize existing networks and drastically reduce the number of bits that encode the weights and inputs of each layer, with only a minimal loss in accuracy and without the need to retrain the networks. With an appropriate hardware architecture this knowledge can be used to reduce energy consumption in many neural network applications and even dynamically scale energy consumption depending on the current application.

Our contribution is threefold:

1. We show how computational precision can be scaled in several ConvNet architectures for image classification through quantization of the layer's inputs and weights. The possible quantization varies per architecture, per application and even per layer within a single ConvNet.
2. We show how this approximate computing / precision scaling can lead to reduced energy consumption in ConvNet accelerators. This is achieved through a combination of algorithmical and circuit-level techniques:
 - Applying precision scaling on ConvNet-accelerator circuits.
 - The number of zero-valued parameter and input-values of ConvNet layers increases through precision scaling. Their computations can be skipped.
3. Based on this analysis of algorithmic accuracy and energy savings, we demonstrate achievable energy-accuracy curves for three popular ConvNet architectures for image classification.

2. Related work

While ConvNets have a long history in the field of computer vision, only recently they have become the go-to technique for classification and detection problems. This newfound popularity is a result of their impressive performance on a number of benchmarks, reaching super-human accuracy on tasks like handwritten character recognition [13] and near-human performance on large scale classification

*Equally contributing authors.

challenges like ILSVRC [4]. Record-breaking ConvNet architectures are being developed in academia [10] [16] and industry [17] [21].

A ConvNet has a hierarchical structure consisting of a number of layers. Most architectures contain multiple blocks of convolutional layers, rectified linear units (ReLU) and pooling layers, followed by a number of fully connected layers. The network is trained with the backpropagation algorithm, which iteratively updates the weights in the convolutional and fully connected layers in order to minimize a certain loss function.

The success of these algorithms has led to the interest of the embedded vision community. A number of ConvNet hardware optimizations [2] [15], coprocessors [6] and accelerators such as DianNao [1] have been proposed. Although all of these works meant a significant step forward, the realization of a high-performance, energy-efficient and real-time ConvNet architecture, which could work autonomously from the cloud is not yet complete. The optimized architectures named above, can however be optimized further by exploiting ConvNet's inherent error-resilience. The effect of quantization errors on regular neural networks have been studied in [8] and [5].

Error resilient applications can be made more energy efficient through approximate computing. This is a collection of hardware-level techniques enabling energy savings at the expense of reduced computational accuracy. This can be achieved in various ways, leading to either a static (fixed after design-time) or a dynamic (adaptable after design-time) trade-off. Static examples in literature make use of approximate digital building blocks for arithmetic functions [11], or use approximate logic synthesis methods [20]. Dynamic approximate computing mainly uses the principle of precision scaling, in which the number of bits used in computations is varied at run-time [18] [3] [14]. In this work, we use precision scaling to achieve energy gains in ConvNet acceleration. Since precision scaling can be applied dynamically, the energy consumption of the targeted accelerator can be adapted to the varying precision requirements of the used ConvNet and even of the ConvNet layer.

Approximate computing through precision scaling lowers the active power consumption of a digital circuit. The total power consumption consists of a dynamic and a leakage component. It can be summarized as $P = \alpha C f V^2 + P_{leakage}$ where α is the circuit switching activity, f is the clock frequency, C is the total switching capacitance and V is the circuit's supply voltage. By scaling precision (i.e., dynamically scaling the number of bits that encode the network's weights and inputs), the switching activity α can be substantially reduced. Figure 1 illustrates the achievable energy-accuracy trade-off in a typical digital multiplier. Note that a striking 12 $\times$ energy gain can be achieved at an output error of 1% root-mean-square-error (RMSE).

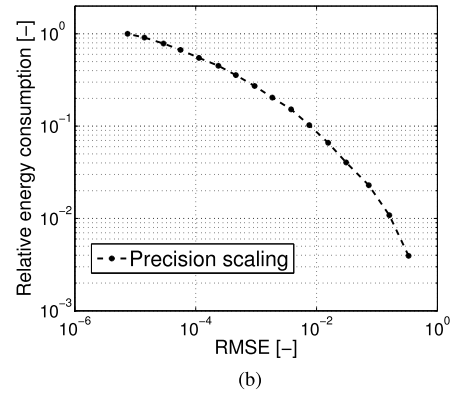
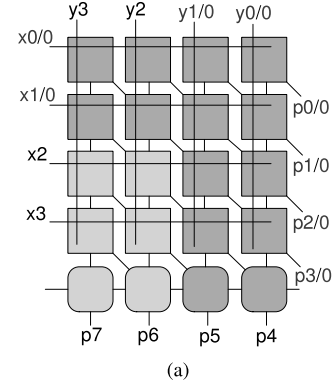


Figure 1: Energy-accuracy trade-off in digital multipliers through precision scaling (a) By scaling the used number of bits from 4 to 2 the number of active blocks drops from 20 to 6. (b) Energy consumption versus root-mean-square-error under precision scaling. Precision scaling can gain 12 $\times$ in energy at 1% RMSE.

A first attempt in combining neural networks with approximate computing has been done in [19]. Their work only looks into regular neural networks and requires retraining of the network. In this work we focus on convolutional neural network architectures, which perform much better in computer vision applications. Furthermore, we do not require network retraining.

3. Analysis and experiments

We examine the performance- and energy-related effects of quantization on three different network architectures for image classification. Because it is impossible to cover the entire spectrum of architectures and applications, we choose three popular networks as representatives of small-, medium-, and large-scale architectures that differ significantly in their number of parameters:

- LeNet-5 on MNIST [12]: LeNet-5 is a small network

with two convolutional and two fully connected layers. It is designed to classify handwritten digits (28x28 images) in the MNIST dataset where it achieves an accuracy of 99.0%.

- **CifarQuick on Cifar-10 [9]:** CifarQuick is a medium-sized architecture with three convolutional and two fully connected layers. It has more filters per layer than LeNet-5 and operates on color images instead of grayscale images. It classifies images of the Cifar-10 dataset, which consists of small 32x32 color images divided into ten classes. It reaches an accuracy of 75.3%.
- **AlexNet on ImageNet [10]:** AlexNet is a large network with five convolutional and three fully connected layers. It reaches a top-5 accuracy of 80.0% on ILSVRC2012, a large scale classification challenge with 1000 categories.

For our experiments we customized the open-source deep learning framework Caffe [7] to be able to simulate quantization of the network's weights and inputs. All experiments are run on the validation sets of the discussed benchmarks. We always report the relative accuracy, i.e. the ratio of the accuracy after quantization and the accuracy of the original network.

Note that our techniques can be used to reduce the energy consumption during evaluation of the network, but not necessarily during the training phase. When training the network, the weight quantization would cause the backpropagation algorithm to quickly get stuck in local optima.

3.1. Influence of quantization on classification accuracy

Typically, ConvNets run on high precision machines using 32-bit floating point number representations. Dedicated embedded platforms use 16-bit fixed point hardware for ConvNet computations. However, such high precision is not always necessary in. The energy spent in high precision computations, does not lead to more accurate classification by the algorithm. To reduce the energy consumption of the ConvNet's computations, our main strategy is to quantize its weights and the inputs to its layers. Such quantization leads to a network that is only an approximation of the original network. Our goal is to find out the influence of quantization on a network's accuracy and whether the effects differ significantly across network architectures. We are interested in how far we can push the quantization of a network without sacrificing too much classification accuracy.

Before quantizing the weights and inputs, it is important to rescale them properly according to the distribution of their values. If there is a mismatch between the interval in which these values lie and the interval over which we quantize, the accuracy will drop even at very low quantization settings. For this reason, we rescale all layer inputs and

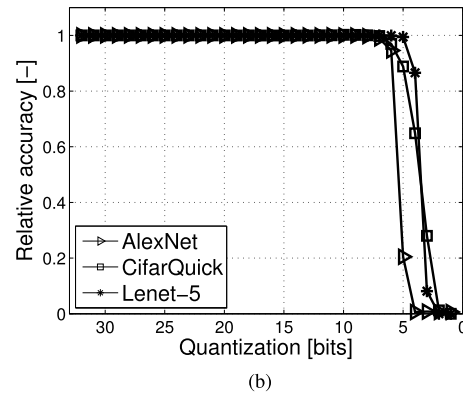
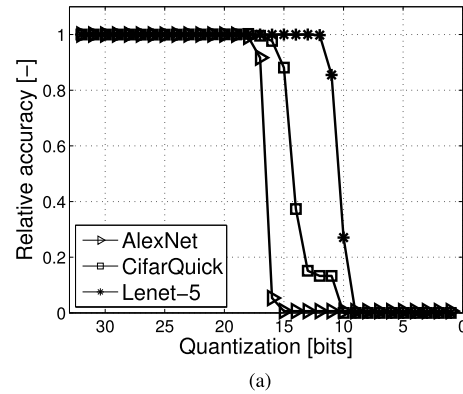


Figure 2: Relative accuracy of three architectures as a function of the number of quantization bits. Layer inputs and weights are first rescaled so they can be quantized more effectively. We compare two rescaling strategies: (a) Uniform rescaling: all inputs and weights are scaled with the same value. (b) Per-layer rescaling: inputs and weights are scaled on a per-layer basis.

weights in the network with a single value that corresponds to the maximal input or weight value (rounded to the next factor of two) observed during a complete run over the data. This ensures that the limits of the quantization interval correspond to the limits of the data, and no bits are wasted. Mathematically, this scaling is the same as multiplying the results with a factor of 2, or shifting data in a fixed point number representation. This operation has a limited hardware energy footprint and only needs to be performed at convolutional outputs.

3.1.1 Uniform quantization

As a first experiment we choose a single quantization setting for all layers in the network, and call this *uniform quantization*. Later, we also introduce *per-layer quantization*, where

each layer is quantized separately.

Figure 2a shows the relative accuracy of our three networks as a function of the number of quantization bits. As can be seen in the figure, the relative accuracy stays equal to one for all three networks up to quantization with 19 bits, meaning that the quantized network reaches the exact same accuracy on its dataset as the original network. At a quantization with 18-bit however, the accuracy for AlexNet starts dropping quickly, rendering it useless for any practical application. At 11-bit the same effect can be seen for the smaller LeNet-5.

We can do better than this by applying the scaling in a more fine-grained way; we choose a different scaling factor for each layer. This is clarified by a simple example: AlexNet's first and sixth layer weight statistics are shown in Figure 3. All weights of layer 1 are within the $[-0.5, 0.5]$ interval, while the weights of layer 6 are within the $[-0.0625, 0.0625]$ interval. By allowing to quantize the weights in layer 6 over this smaller interval instead of over $[-0.5, 0.5]$ we recover 3 bits that would otherwise be wasted. A similar reasoning holds for the layer's inputs.

The effect of the per-layer scaling is significant, as can be seen in Figure 2b. Compared to the previous, we can now quantize much more aggressively without sacrificing accuracy. Each network performs at its original accuracy up to quantization with no more than 8 bits. After that, the accuracy again drops quickly. The reason for this improvement is the fact that input and weight statistics differ greatly among layers. Per-layer rescaling allows to set the optimal quantization interval in each layer.

3.1.2 Per-layer quantization

Just as we do per-layer rescaling, we can also do per-layer quantization: instead of quantizing all weights and inputs of the network with the same number of bits, we choose a different setting in each layer. The idea is again to find an optimal setting for each layer by exploiting the variations of the layer-specific input and weight distributions. Another effect of the per-layer quantization is that we can set the operating point (i.e. the desired minimal relative accuracy) more precisely and thus control the energy-accuracy trade-off tightly, as discussed later in section 3.3. This is not possible with uniform quantization: e.g. between quantization with 5 and 4 bits, the relative accuracy of LeNet-5 drops from 99.4% immediately to an unusable 86.6%.

In order to find a good quantization setting for each layer, we do a greedy search over the parameters: starting at the first layer, its input is quantized until the accuracy drops to the target accuracy. Next, the quantization of the input is kept fixed while the quantization of its weights is maximized in the same way. The same process is applied in the next layers until the last one. If a value goes out of range,

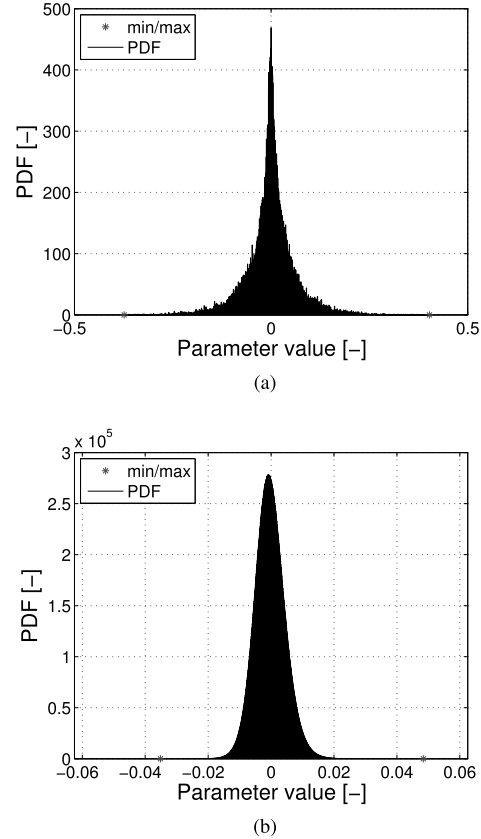


Figure 3: AlexNet's first and sixth layers weight statistics. Layer 1 should be projected on the $[-0.5, 0.5]$ interval and layer 6 on the $[-0.0625, 0.0625]$ interval to minimize the needed number of bits. If layer 6 would be projected on the $[-1, 1]$ interval, 4 extra bits would be needed.

we clip it to the respective minimum or maximum representable value with the chosen quantization. The full sweep can take many hours, depending on the network size, dataset size and number of tested settings. This poses no real problem, as the procedure has to be performed only once and can be done off-line.

The amount of bits that can be saved with per-layer quantization at a target accuracy of 99%, compared to uniform quantization at 100% relative accuracy, is visualized for each reference network in Figure 4. The results are ad-hoc, but there is a general trend of needing less bits in the lower layers of the network than in the higher layers. This is partly a result of the forward parameter sweep, but we hypothesize that the difference in input and weight statistics between lower and higher layers also plays a significant role.

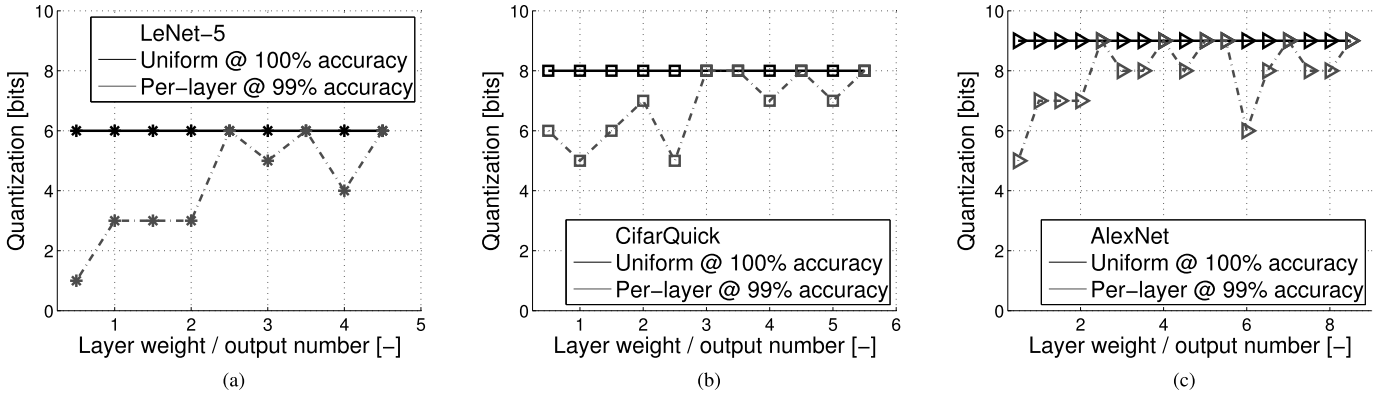


Figure 4: Comparison of uniform and optimal per-layer quantization. Data points at integer values show quantization of the layer weights. Other data points show quantization of the layer inputs. Maximum achievable quantizations: (a) LeNet-5: 1-6 bit. (b) CifarQuick: 5-8 bit. (c) AlexNet: 5-9 bit.

3.2. Proposed energy reduction through precision scaling and computation skipping

This section indicates how this increased quantization can lead to energy savings in real hardware architectures. To quantify the possible energy gains of approximate computing in ConvNet acceleration, we model the energy consumption of the necessary convolutional arithmetic (multiply and add) for a full ConvNet algorithm. This analysis does not incorporate the energy overheads of control, I/O, data- and program-memory interfacing and the clock network, as these are highly architecture dependent. It can thus be considered as the maximum potential energy savings through approximate computing. Furthermore, the number of zero-valued weights and inputs of typical ConvNets increases at higher quantization. This can be exploited algorithmically by skipping unnecessary computations with zero-valued inputs. This will lead to large additional energy savings.

3.2.1 Precision scaling

Precision scaling is an approximate computing technique allowing major energy savings in digital circuits. Lower precision computing, i.e. computing using less bits, reduces the switching activity. Figure 1 and [14] show how this concept can be applied to a common digital multiplier.

Based on simulations in a 40nm technology, an energy model for common digital building blocks is proposed in Table 1. The reduced switching activity, and hence energy consumption, solely depends on the circuit architecture and can reduce quadratically (multiplier), or linearly (adders, register files, wiring). Using the energy-modelling from Ta-

Table 1: First order estimation of power consumption in digital basic building blocks under accuracy scaling, based on simulations in a 40nm CMOS technology. The modelled blocks are multipliers ($\times$), adders ($+$), registers, wires and SRAM memory. In the power equation (P), only the switching activity α decreases as a function of the used number of bits n .

$P = \alpha C f V_{dd}^2$	$\times$	$+$	Reg.	Wire	SRAM
α	$O(n^2)$	$O(n)$	$O(n)$	$O(n)$	$O(n_{max})$

ble 1, we can estimate the impact of increased quantization arithmetic on the energy consumption of a full ConvNet-algorithms.

In order to apply precision scaling efficiently in such a system, architectural changes have to be made. First, the positioning of the sign-bit in the fixed number representation is crucial. The MSB-bit should always be placed at the MSB position, otherwise toggling sign bits will lead to high switching activity. Second, The number of bits in a fixed point implementation tends to expand due to numerical operations. In order to make sure the number of bits remains limited throughout different stages of the algorithm, repetitive rounding or truncation is needed after every stage. Third, data- and parameter rescaling should be implemented between the convolutional layers using arithmetic shifting.

3.2.2 Computation skipping

An interesting feature of many modern convolutional neural networks is the appearance of Rectified Linear Unit

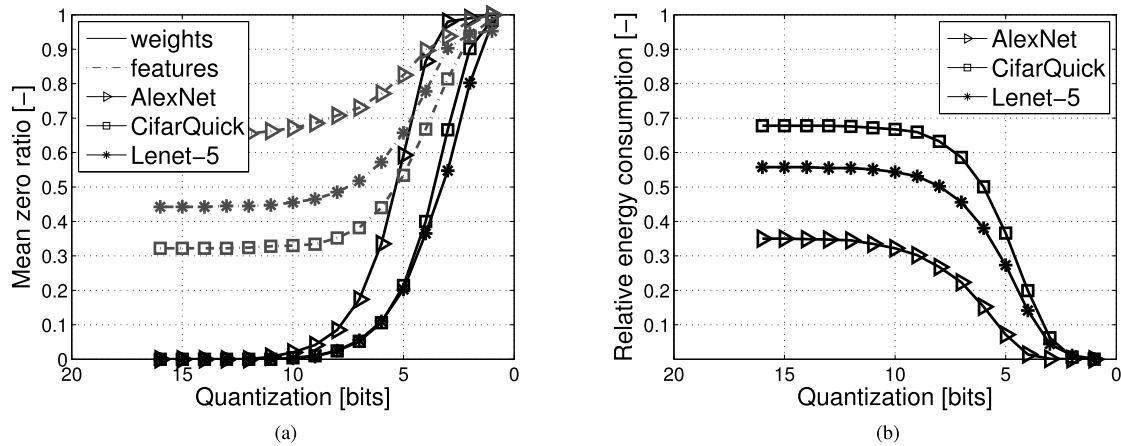


Figure 5: (a) Impact of precision scaling on the average number of zeroes in the weights and input values of all three examples. (b) Relative energy reduction compared to the case without computation skipping.

layers (ReLU layers). These put all negative inputs to zero and pass on positive values unchanged, as in $Output = \max(0, Input)$. Since many layers in ConvNet-classification algorithms only output positive values when certain features are present, a large amount of ReLU-outputs will be zero and do not have to be used for further computations. The ReLU layers thus allow for additional energy reductions by not computing unnecessary computations through computation skipping.

Figure 5 shows the impact of precision scaling on the average number of zeroes in our ConvNet examples. For all architectures, the ratio of zeroed values lies between 50 – 90% of the total input values, depending on the used quantization setting. The number of zeroed values is significantly larger under precision scaling, from an average of 45% on all LeNet-5 outputs at 16-bit to 65% at 5-bit.

It is very difficult to achieve such computation skipping in a software solution, since checking for zero-valued inputs in a scalar core consumes time. Using techniques for sparse matrix multiplication is not possible either, because the used convolutional kernels are typically very small on the order of 11×11 , 5×5 or 3×3 . However, computation skipping can be achieved in a hardware accelerator with dedicated hardware support. Flags can indicate if upcoming data is zero and prevent circuitry from switching if this is the case. As ConvNet-weights are fixed, these flags can be computed beforehand.

3.3. Achievable energy-accuracy trade-off in ConvNets

By combining both precision scaling and algorithmical computation skipping, major energy savings can be achieved in ConvNets. Figure 6a shows the effect of uni-

form quantization on the energy consumption of our benchmark algorithms, if both precision scaling and computation skipping are combined. Note how the energy reduces 5 – 20× for an 8-bit implementation.

Figure 6b shows curves in the energy-accuracy space for our ConvNet benchmarks. It shows the trade-off for the 95 – 100% classification accuracy window. All curves are given for both uniform and per-layer quantization, and are compared to the typically used uniform 16-bit number representation, without adequate precision scaling and computation skipping. If per-layer quantization is used, the trade-off is less steep than in the uniform case. This means more energy can be gained, while losing less classification accuracy in the process. All discussed networks gain at least an additional factor of 2 in energy consumption if a reduced classification accuracy of 99% is allowed.

Figure 7 gives an overview of the effect of each improvement on the relative energy consumption. Per-layer rescaling and computation skipping lead to the largest energy gains. Per-layer quantization leads to additional major energy gains if a reduced accuracy of 99% can be allowed.

4. Conclusion

ConvNet algorithms are typically very computation and memory intensive. In order to be able to embed ConvNet-based classification into wearable platforms, their energy consumption should be reduced significantly. In this work, we show how the energy consumption of dedicated ConvNet accelerators can be drastically reduced by applying approximate computing. ConvNets for image classification are fault-tolerant and allow computation using 4-10 bits instead of the previously used 16- or 32-bit formats. By us-

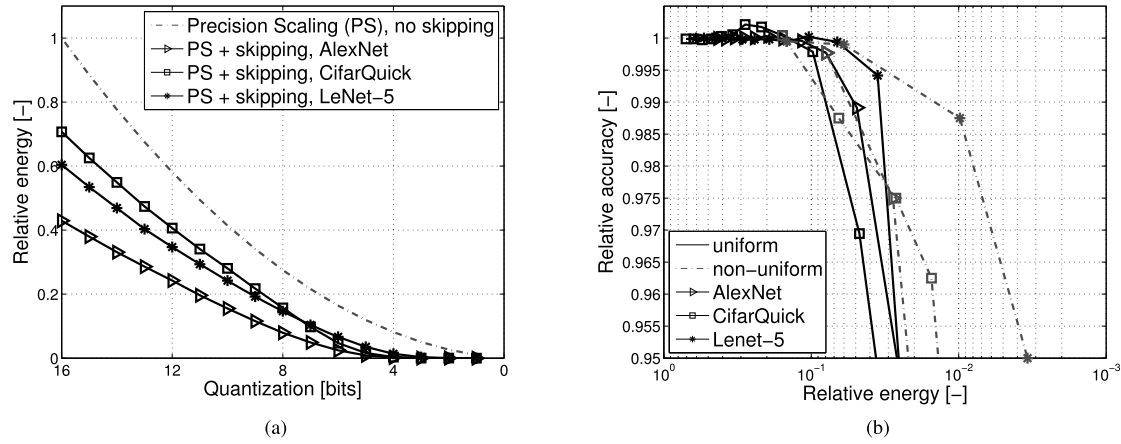


Figure 6: (a) Relative energy consumption for uniform quantization. A striking $5 - 20\times$ lower energy consumption at 8-bit, compared to the commonly used 16-bit used in previous literature. (b) Energy-accuracy plot for per-layer quantization and scaling and with computation skipping. All considered ConvNet-networks gain at least an additional factor of 2 in energy consumption if 99% accuracy is allowed.

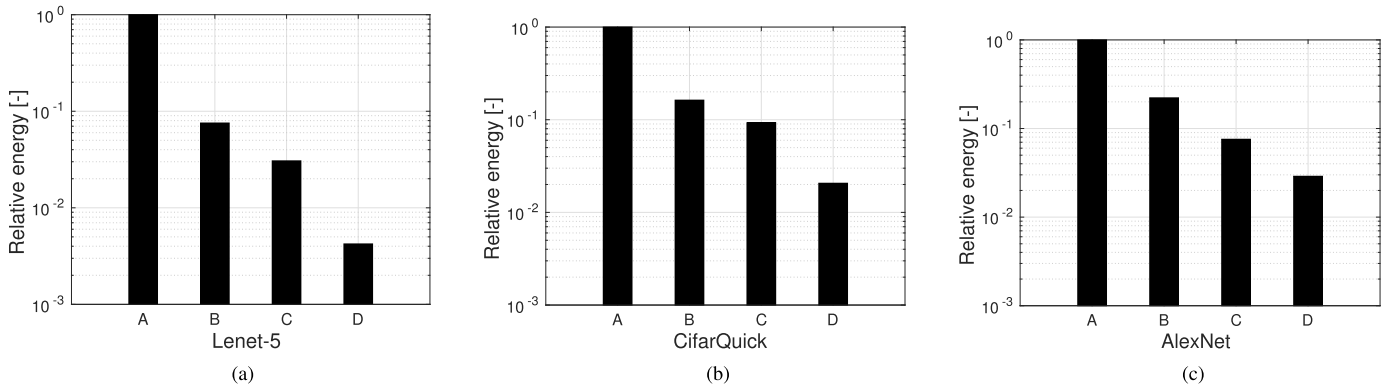


Figure 7: Energy comparison between our three benchmarks in different cases. (A) 16-bit (B) Per-layer scaling, uniform quantization at 100% accuracy (C) Same as B, with computation skipping (D) Per-layer scaling, per-layer quantization and computation skipping at 99% accuracy.

ing requantized number representations, the energy consumption in a ConvNet accelerator can be reduced in two complementary ways. First, energy consumption can be lower through precision scaling. Second, more quantization further increases the number of zeroed parameter and input values in the ConvNet. Since zero-valued numbers do not contribute to the algorithm's output, their computations can be skipped. The combination of these two techniques can lead to significant energy savings in several ConvNets. We show energy reductions of up to $30\times$ compared to the commonly used 16-bit fixed point implementations, with-

out sacrificing algorithm performance. If the classification accuracy can be reduced to 99%, additional energy savings can be achieved on the same network architecture, through per-layer optimization. In this case an additional $7.5\times$ reduction can be achieved.

References

- [1] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. In *Proceedings of the 19th international conference on Architectural support*

- for programming languages and operating systems, pages 269–284. ACM, 2014. 2
- [2] F. Conti and L. Benini. A ultra-low-energy convolution engine for fast brain-inspired vision in multicore clusters. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, pages 683–688. EDA Consortium, 2015. 2
- [3] M. de la Guia Solaz, W. Han, and R. Conway. A flexible low power dsp with a programmable truncated multiplier. *Circuits and Systems I: Regular Papers, IEEE Transactions on*, 59(11):2555–2568, 2012. 2
- [4] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition, 2009. CVPR 2009*, pages 248–255, June 2009. 2
- [5] G. Dunder and K. Rose. The effects of quantization on multilayer neural networks. *IEEE transactions on neural networks/a publication of the IEEE Neural Networks Council*, 6(6):1446–1451, 1994. 2
- [6] C. Farabet, B. Martini, B. Corda, P. Akselrod, E. Culurciello, and Y. LeCun. Neuflow: A runtime reconfigurable dataflow processor for vision. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2011 IEEE Computer Society Conference on*, pages 109–116. IEEE, 2011. 2
- [7] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional Architecture for Fast Feature Embedding. *arXiv:1408.5093 [cs]*, June 2014. arXiv: 1408.5093. 3
- [8] M. Jiang and G. Gielen. The effects of quantization on multilayer feedforward neural networks. *International Journal of Pattern Recognition and Artificial Intelligence*, 17(04):637–661, June 2003. 2
- [9] A. Krizhevsky. Learning multiple layers of features from tiny images. 2009. 3
- [10] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. 2, 3
- [11] P. Kulkarni, P. Gupta, and M. Ercegovac. Trading accuracy for power with an underdesigned multiplier architecture. In *VLSI Design (VLSI Design), 2011 24th International Conference on*, pages 346–351. IEEE, 2011. 2
- [12] B. B. Le Cun, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*. Citeseer, 1990. 2
- [13] Y. LeCun and C. Cortes. The mnist database of handwritten digits, 1998. 1
- [14] B. Moons and M. Verhelst. Dvas: Dynamic voltage accuracy scaling for increased energy-efficiency in approximate computing. *International Symposium on Low Power Electronics and Design (ISLPED)*, 2015. 2, 5
- [15] M. Peemen, A. A. Setio, B. Mesman, and H. Corporaal. Memory-centric accelerator design for convolutional neural networks. In *Computer Design (ICCD), 2013 IEEE 31st International Conference on*, pages 13–19. IEEE, 2013. 2
- [16] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. 2
- [17] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going Deeper with Convolutions. *arXiv:1409.4842 [cs]*, Sept. 2014. arXiv: 1409.4842. 2
- [18] S. Venkataramani, V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan. Quality programmable vector processors for approximate computing. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 1–12. ACM, 2013. 2
- [19] S. Venkataramani, A. Ranjan, K. Roy, and A. Raghunathan. Axnn: energy-efficient neuromorphic systems using approximate computing. In *Proceedings of the 2014 international symposium on Low power electronics and design*, pages 27–32. ACM, 2014. 2
- [20] S. Venkataramani, A. Sabne, V. Kozhikkottu, K. Roy, and A. Raghunathan. Salsa: systematic logic synthesis of approximate circuits. In *Proceedings of the 49th Annual Design Automation Conference*, pages 796–801. ACM, 2012. 2
- [21] R. Wu, S. Yan, Y. Shan, Q. Dang, and G. Sun. Deep Image: Scaling up Image Recognition. *arXiv:1501.02876 [cs]*, Jan. 2015. arXiv: 1501.02876. 2

ABSTRACT B

Energy-efficient ConvNets through approximate computing

Publisher: IEEE

Cite This

Bert Moons ; Bert De Brabandere ; Luc Van Gool ; Marian Verhelst **All Authors**
78
Cites in
Papers

7
Cites in
Patents

1713
Full
Text Views


Abstract

Document Sections

1. Introduction
2. Related Work
3. Analysis and Experiments
4. Conclusion

Authors

Figures

References

Citations

Keywords

Metrics

More Like This

Abstract:

Recently convolutional neural networks (ConvNets) have come up as state-of-the-art classification and detection algorithms, achieving near-human performance in visual detection. However, ConvNet algorithms are typically very computation and memory intensive. In order to be able to embed ConvNet-based classification into wearable platforms and embedded systems such as smartphones or ubiquitous electronics for the internet-of-things, their energy consumption should be reduced drastically. This paper proposes methods based on approximate computing to reduce energy consumption in state-of-the-art ConvNet accelerators. By combining techniques both at the system- and circuit level, we can gain energy in the systems arithmetic: up to 30× without losing classification accuracy and more than 100× at 99% classification accuracy, compared to the commonly used 16-bit fixed point number format.

Published in: 2016 IEEE Winter Conference on Applications of Computer Vision (WACV)

Date of Conference: 07-10 March 2016

DOI: 10.1109/WACV.2016.7477614

Date Added to IEEE Xplore: 26 May 2016

Publisher: IEEE

► **ISBN Information:**

Conference Location: Lake Placid, NY, USA

SECTION 1. Introduction

Recently neural networks have made an impressive comeback in the field of machine learning. Convolutional neural networks or ConvNets are consistently pushing the state-of-the-art in areas like computer vision and speech processing. One of the reasons for this revival is the increasing availability of computing power. Multicore CPU's, GPU's, and even clusters of GPU's are no longer prohibitively expensive and make it possible to train and evaluate larger networks.

Unfortunately, the increase in computing power is inevitably accompanied by an increase in energy consumption. While high energy consumption is no big concern during the network's training phase - which typically takes place on a computer cluster - it poses a problem when the network needs to be evaluated on mobile hardware like smartphones, smart glasses and other wearable devices.

In this paper, we investigate how energy consumption can be reduced through principles of approximate computing. Our analysis shows that it is possible to quantize existing networks and drastically reduce the number of bits that encode the weights and inputs of each layer, with only a minimal loss in accuracy and without the need to retrain the networks. With an appropriate hardware architecture this knowledge can be used to reduce energy consumption in many neural network applications and even dynamically scale energy consumption depending on the current application.

Our contribution is threefold:

1. We show how computational precision can be scaled in several ConvNet architectures for image classification through quantization of the layer's inputs and weights. The possible quantization varies per



architecture, per application and even per layer within a single ConvNet.

2. We show how this approximate computing/precision scaling can lead to reduced energy consumption in ConvNet accelerators. This is achieved through a combination of algorithmic and circuit-level techniques:

- Applying precision scaling on ConvNet-accelerator circuits.
 - The number of zero-valued parameter and input-values of ConvNet layers increases through precision scaling. Their computations can be skipped.
3. Based on this analysis of algorithmic accuracy and energy savings, we demonstrate achievable energy-accuracy curves for three popular ConvNet architectures for image classification.

SECTION 2. Related Work

While ConvNets have a long history in the field of computer vision, only recently they have become the go-to technique for classification and detection problems. This newfound popularity is a result of their impressive performance on a number of benchmarks, reaching super-human accuracy on tasks like handwritten character recognition [13] and near-human performance on large scale classification challenges like ILSVRC [4]. Record-breaking ConvNet architectures are being developed in academia [10] [16] and industry [17] [21].

A ConvNet has a hierarchical structure consisting of a number of layers. Most architectures contain multiple blocks of convolutional layers, rectified linear units (ReLU) and pooling layers, followed by a number of fully connected layers. The network is trained with the backpropagation algorithm, which iteratively updates the weights in the convolutional and fully connected layers in order to minimize a certain loss function.

The success of these algorithms has led to the interest of the embedded vision community. A number of ConvNet hardware optimizations [2] [15], coprocessors [6] and accelerators such as DianNao [1] have been proposed. Although all of these works meant a significant step forward, the realization of a high-performance, energy-efficient and real-time ConvNet architecture, which could work autonomously from the cloud is not yet complete. The optimized architectures named above, can however be optimized further by exploiting ConvNet's inherent error-resilience. The effect of quantization errors on regular neural networks have been studied in [8] and [5].

Error resilient applications can be made more energy efficient through approximate computing. This is a collection of hardware-level techniques enabling energy savings at the expense of reduced computational accuracy. This can be achieved in various ways, leading to either a static (fixed after design-time) or a dynamic (adaptable after design-time) trade-off. Static examples in literature make use of approximate digital building blocks for arithmetic functions [11], or use approximate logic synthesis methods [20]. Dynamic approximate computing mainly uses the principle of precision scaling, in which the number of bits used in computations is varied at run-time [18] [3] [14]. In this work, we use precision scaling to achieve energy gains in ConvNet acceleration. Since precision scaling can be applied dynamically, the energy consumption of the targeted accelerator can be adapted to the varying precision requirements of the used ConvNet and even of the ConvNet layer.

Approximate computing through precision scaling lowers the active power consumption of a digital circuit. The total power consumption consists of a dynamic and a leakage component. It can be summarized as $P = \alpha C f V^2 + P_{leakage}$ where α is the circuit switching activity, f is the clock frequency, C is the total switching capacitance and V is the circuit's supply voltage. By scaling precision (i.e., dynamically scaling the number of bits that encode the network's weights and inputs), the switching activity α can be substantially reduced. Figure 1 illustrates the achievable energy-accuracy trade-off in a typical digital multiplier. Note that a striking 12× energy gain can be achieved at an output error of 1% root-mean-square-error (RMSE).

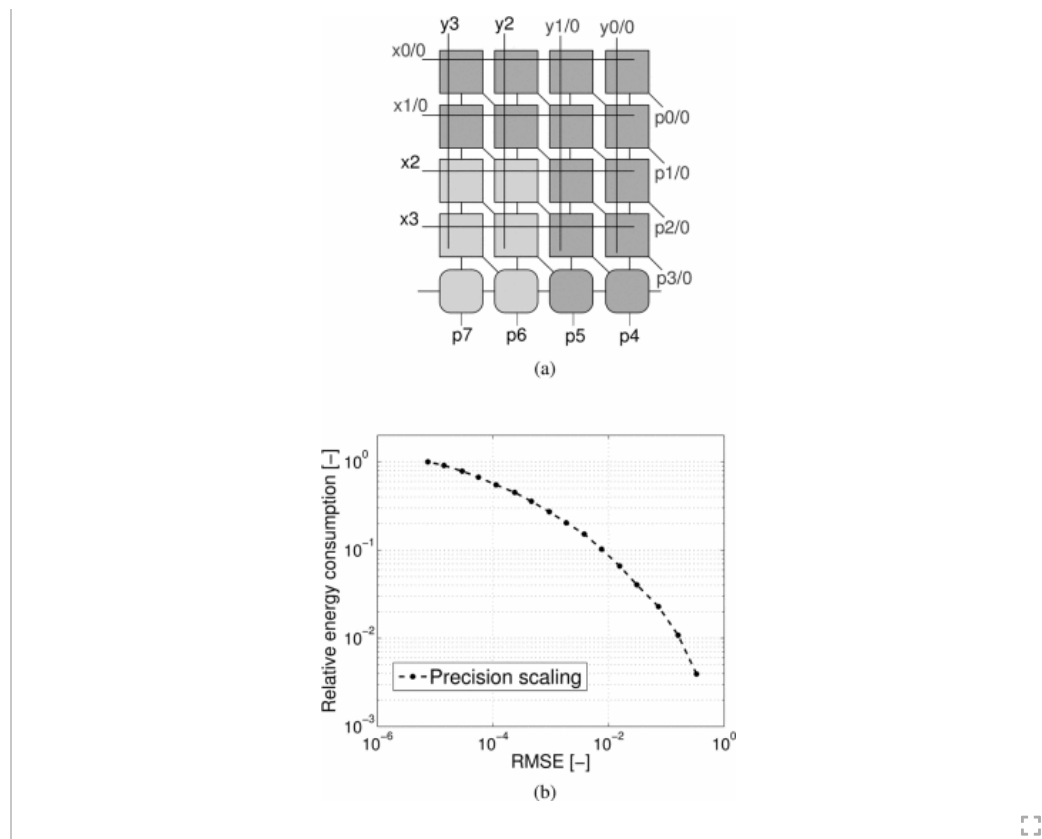


Figure 1:

Energy-accuracy trade-off in digital multipliers through precision scaling (a) by scaling the used number of bits from 4 to 2 the number of active blocks drops from 20 to 6. (b) Energy consumption versus root-mean-square-error under precision scaling. Precision scaling can gain $12\times$ in energy at 1% rmse.

A first attempt in combining neural networks with approximate computing has been done in [19]. Their work only looks into regular neural networks and requires retraining of the network. In this work we focus on convolutional neural network architectures, which perform much better in computer vision applications. Furthermore, we do not require network retraining.

SECTION 3. Analysis and Experiments

We examine the performance-and energy-related effects of quantization on three different network architectures for image classification. Because it is impossible to cover the entire spectrum of architectures and applications, we choose three popular networks as representatives of small, medium-, and large-scale architectures that differ significantly in their number of parameters:

- **LeNet-5 on MNIST [12]:** LeNet-5 is a small network with two convolutional and two fully connected layers. It is designed to classify handwritten digits (28×28 images) in the MNIST dataset where it achieves an accuracy of 99.0%.
- **CifarQuick on Cifar-10 [9]:** CifarQuick is a medium-sized architecture with three convolutional and two fully connected layers. It has more filters per layer than LeNet-5 and operates on color images instead of grayscale images. It classifies images of the Cifar-10 dataset, which consists of small 32×32 color images divided into ten classes. It reaches an accuracy of 75.3%.

- AlexNet on ImageNet [10]: AlexNet is a large network with five convolutional and three fully connected layers. It reaches a top-5 accuracy of 80.0% on ILSVRC2012, a large scale classification challenge with 1000 categories.

For our experiments we customized the open-source deep learning framework Caffe [7] to be able to simulate quantization of the network's weights and inputs. All experiments are run on the validation sets of the discussed benchmarks. We always report the relative accuracy, i.e. the ratio of the accuracy after quantization and the accuracy of the original network.

Note that our techniques can be used to reduce the energy consumption during evaluation of the network, but not necessarily during the training phase. When training the network, the weight quantization would cause the backpropagation algorithm to quickly get stuck in local optima.

3.1. Influence of Quantization on Classification Accuracy

Typically, ConvNets run on high precision machines using 32-bit floating point number representations. Dedicated embedded platforms use 16-bit fixed point hardware for ConvNet computations. However, such high precision is not always necessary in. The energy spent in high precision computations, does not lead to more accurate classification by the algorithm. To reduce the energy consumption of the ConvNet's computations, our main strategy is to quantize its weights and the inputs to its layers. Such quantization leads to a network that is only an approximation of the original network. Our goal is to find out the influence of quantization on a network's accuracy and whether the effects differ significantly across network architectures. We are interested in how far we can push the quantization of a network without sacrificing too much classification accuracy.

Before quantizing the weights and inputs, it is important to rescale them properly according to the distribution of their values. If there is a mismatch between the interval in which these values lie and the interval over which we quantize, the accuracy will drop even at very low quantization settings. For this reason, we rescale all layer inputs and weights in the network with a single value that corresponds to the maximal input or weight value (rounded to the next factor of two) observed during a complete run over the data. This ensures that the limits of the quantization interval correspond to the limits of the data, and no bits are wasted. Mathematically, this scaling is the same as multiplying the results with a factor of 2, or shifting data in a fixed point number representation. This operation has a limited hardware energy footprint and only needs to be performed at convolutional outputs.

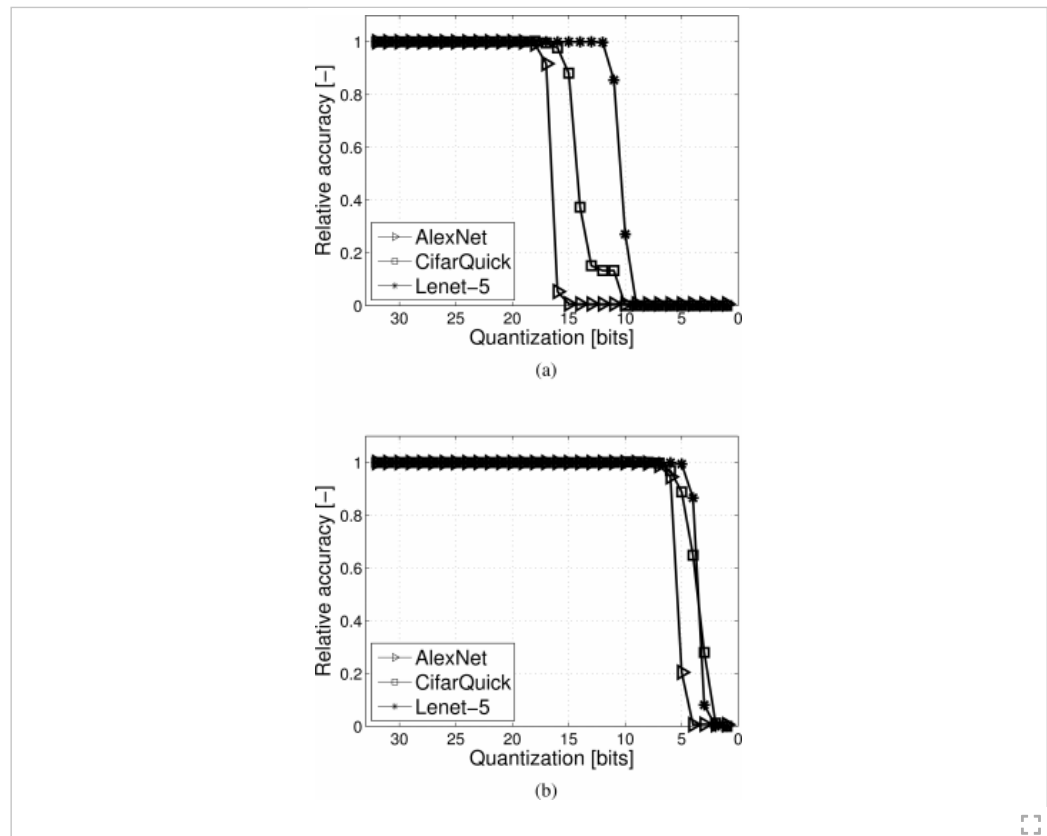


Figure 2:

Relative accuracy of three architectures as a function of the number of quantization bits. Layer inputs and weights are first rescaled so they can be quantized more effectively. We compare two rescaling strategies: (a) Uniform rescaling: all inputs and weights are scaled with the same value. (b) Per-layer rescaling: inputs and weights are scaled on a per-layer basis.

3.1.1 Uniform Quantization

As a first experiment we choose a single quantization setting for all layers in the network, and call this *uniform quantization*. Later, we also introduce *per-layer quantization*, where each layer is quantized separately.

[Figure 2a](#) shows the relative accuracy of our three networks as a function of the number of quantization bits. As can be seen in the figure, the relative accuracy stays equal to one for all three networks up to quantization with 19 bits, meaning that the quantized network reaches the exact same accuracy on its dataset as the original network. At a quantization with 18-bit however, the accuracy for AlexNet starts dropping quickly, rendering it useless for any practical application. At 11-bit the same effect can be seen for the smaller LeNet-5.

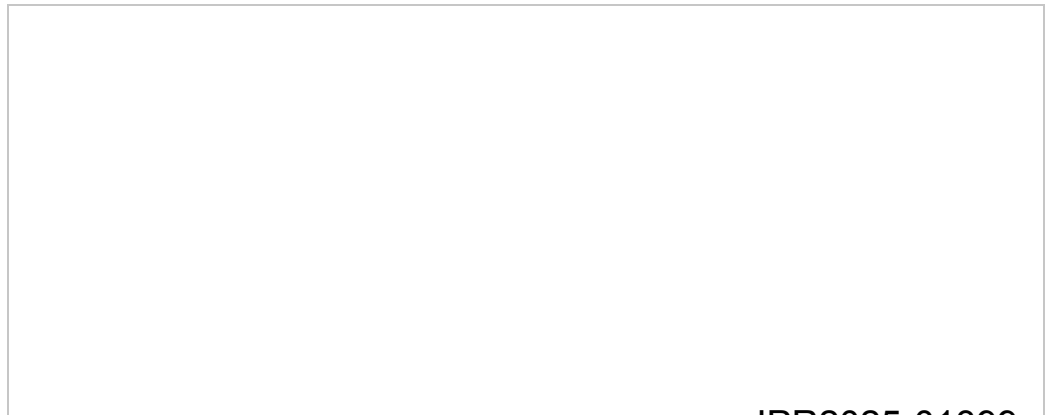
We can do better than this by applying the scaling in a more fine-grained way; we choose a different scaling factor for each layer. This is clarified by a simple example: AlexNet's first and sixth layer weight statistics are shown in [Figure 3](#). All weights of layer 1 are within the $[-0.5, 0.5]$ interval, while the weights of layer 6 are within the $[-0.0625, 0.0625]$ interval. By allowing to quantize the weights in layer 6 over this smaller interval instead of over $[-0.5, 0.5]$ we recover 3 bits that would otherwise be wasted. A similar reasoning holds for the layer's inputs.

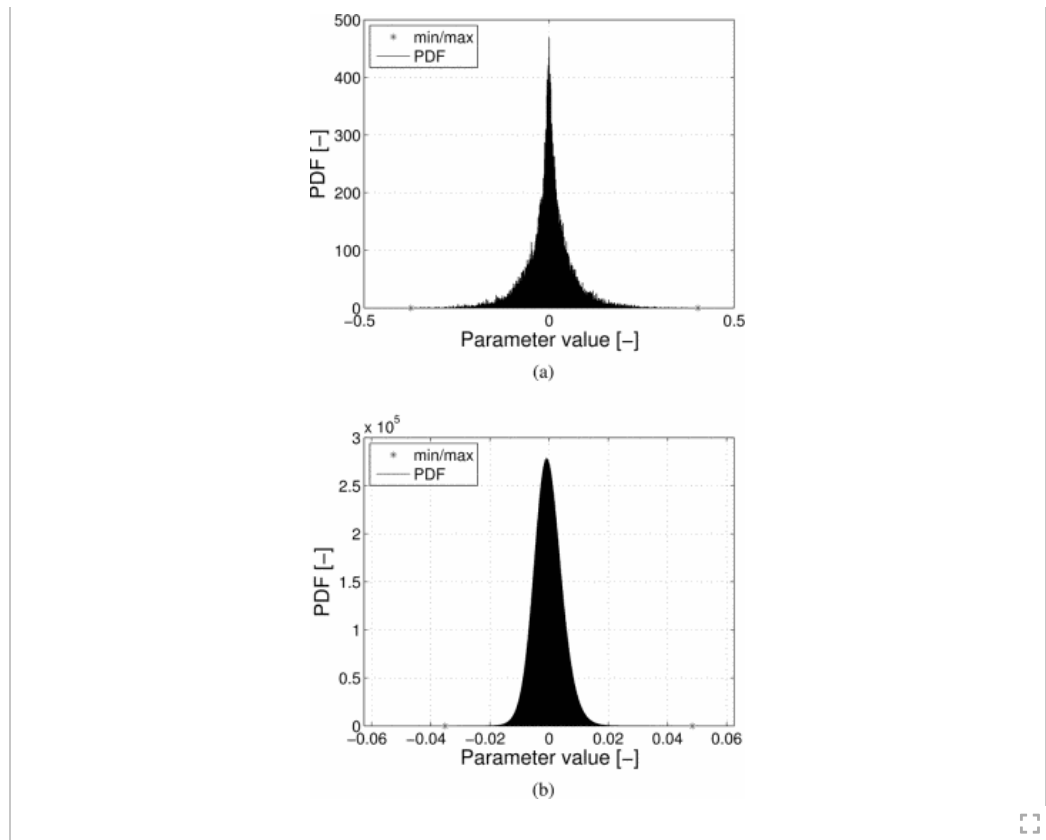
The effect of the per-layer scaling is significant, as can be seen in [Figure 2b](#). Compared to the previous, we can now quantize much more aggressively without sacrificing accuracy. Each network performs at its original accuracy up to quantization with no more than 8 bits. After that, the accuracy again drops quickly. The reason for this improvement is the fact that input and weight statistics differ greatly among layers. Per-layer rescaling allows to set the optimal quantization interval in each layer.

3.1.2 Per-Layer Quantization

Just as we do per-layer rescaling, we can also do per-layer quantization: instead of quantizing all weights and inputs of the network with the same number of bits, we choose a different setting in each layer. The idea is again to find an optimal setting for each layer by exploiting the variations of the layer-specific input and weight distributions. Another effect of the per-layer quantization is that we can set the operating point (i.e. the desired minimal relative accuracy) more precisely and thus control the energy-accuracy trade-off tightly, as discussed later in [section 3.3](#). This is not possible with uniform quantization: e.g. between quantization with 5 and 4 bits, the relative accuracy of LeNet-5 drops from 99.4% immediately to an unusable 86.6%.

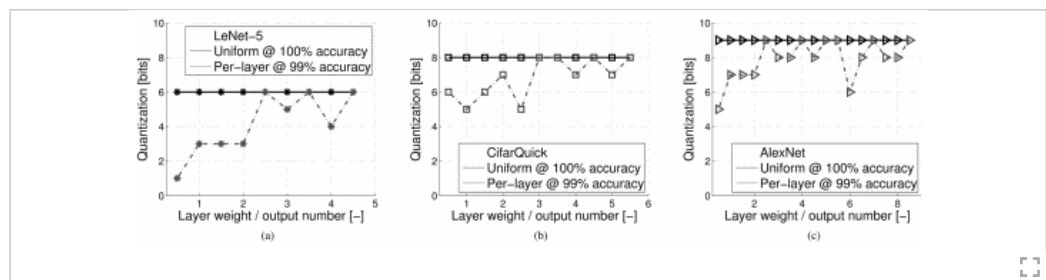
In order to find a good quantization setting for each layer, we do a greedy search over the parameters: starting at the first layer, its input is quantized until the accuracy drops to the target accuracy. Next, the quantization of the input is kept fixed while the quantization of its weights is maximized in the same way. The same process is applied in the next layers until the last one. If a value goes out of range, we clip it to the respective minimum or maximum representable value with the chosen quantization. The full sweep can take many hours, depending on the network size, dataset size and number of tested settings. This poses no real problem, as the procedure has to be performed only once and can be done off-line.



**Figure 3:**

Alexnet's first and sixth layers weight statistics. Layer 1 should be projected on the $[-0.5, 0.5]$ interval and layer 6 on the $[-0.0625, 0.0625]$ interval to minimize the needed number of bits. If layer 6 would be projected on the $[-1, 1]$ interval, 4 extra bits would be needed.

The amount of bits that can be saved with per-layer quantization at a target accuracy of 99%, compared to uniform quantization at 100% relative accuracy, is visualized for each reference network in Figure 4. The results are ad-hoc, but there is a general trend of needing less bits in the lower layers of the network than in the higher layers. This is partly a result of the forward parameter sweep, but we hypothesize that the difference in input and weight statistics between lower and higher layers also plays a significant role.

**Figure 4:**

Comparison of uniform and optimal per-layer quantization. Data points at integer values show quantization of the layer weights. Other data points show quantization of the layer inputs. Maximum achievable quantizations: (a) Lenet-5: 1–6 bit. (b) Cifarquick: 5–8 bit. (c) Alexnet: 5–9 bit.

3.2. Proposed Energy Reduction through Precision Scaling and Computation Skipping

This section indicates how this increased quantization can lead to energy savings in real hardware architectures. To quantify the possible energy gains of approximate computing in ConvNet acceleration, we model the energy consumption of the necessary convolutional arithmetic (multiply and add) for a full ConvNet algorithm. This analysis does not incorporate the energy overheads of control, I/O, data- and program-memory interfacing and the clock network, as these are highly architecture dependent. It can thus be

considered as the maximum potential energy savings through approximate computing. Furthermore, the number of zero-valued weights and inputs of typical ConvNets increases at higher quantization. This can be exploited algorithmically by skipping unnecessary computations with zero-valued inputs. This will lead to large additional energy savings.

3.2.1 Precision Scaling

Precision scaling is an approximate computing technique allowing major energy savings in digital circuits. Lower precision computing, i.e. computing using less bits, reduces the switching activity. Figure 1 and [14] show how this concept can be applied to a common digital multiplier.

Based on simulations in a 40nm technology, an energy model for common digital building blocks is proposed in Table 1. The reduced switching activity, and hence energy consumption, solely depends on the circuit architecture and can reduce quadratically (multiplier), or linearly (adders, register files, wiring). Using the energy-modelling from Table 1, we can estimate the impact of increased quantization arithmetic on the energy consumption of a full ConvNet-algorithms.

Table 1: First order estimation of power consumption in digital basic building blocks under accuracy scaling, based on simulations in a 40nm CMOS technology. The modelled blocks are multipliers ($\times$), adders ($+$), registers, wires and SRAM memory. In the power equation (P), only the switching activity α decreases as a function of the used number of bits n

$P = \alpha C f V_{dd}^2$	$\times$	$+$	Reg.	Wire	SRAM
α	$O(n^2)$	$O(n)$	$O(n)$	$O(n)$	$O(n_{max})$

In order to apply precision scaling efficiently in such a system, architectural changes have to be made. First, the positioning of the sign-bit in the fixed number representation is crucial. The MSB-bit should always be placed at the MSB position, otherwise toggling sign bits will lead to high switching activity. Second, The number of bits in a fixed point implementation tends to expand due to numerical operations. In order to make sure the number of bits remains limited throughout different stages of the algorithm, repetitive rounding or truncation is needed after every stage. Third, data-and parameter rescaling should be implemented between the convolutional layers using arithmetic shifting.

3.2.2 Computation Skipping

An interesting feature of many modern convolutional neural networks is the appearance of Rectified Linear Unit layers (ReLU layers). These put all negative inputs to zero and pass on positive values unchanged, as in $Output = \max(0, Input)$. Since many layers in ConvNet-classification algorithms only output positive values when certain features are present, a large amount of ReLU-outputs will be zero and do not have to be used for further computations. The ReLU layers thus allow for additional energy reductions by not computing unnecessary computations through computation skipping.

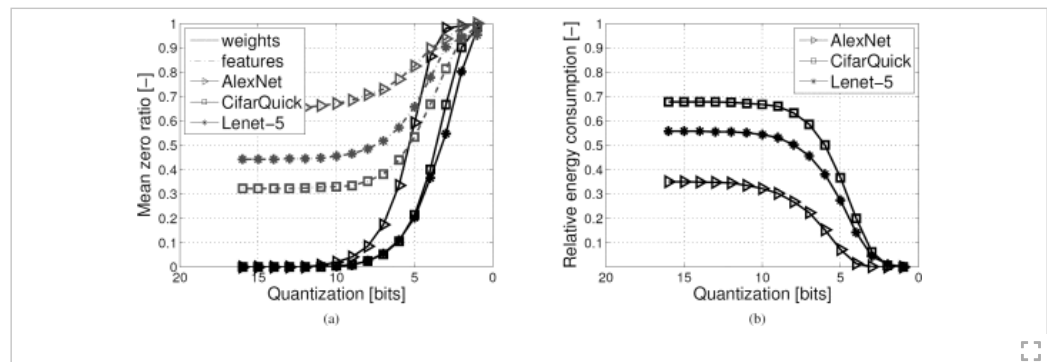


Figure 5:

(a) Impact of precision scaling on the average number of zeroes in the weights and input values of all three examples. (b) Relative energy reduction compared to the case without computation skipping.

Figure 5 shows the impact of precision scaling on the average number of zeroes in our ConvNet examples. For all architectures, the ratio of zeroed values lies between 50-90% of the total input values, depending on the used quantization setting. The number of zeroed values is significantly larger under precision scaling, from an average of 45% on all LeNet-5 outputs at 16-bit to 65% at 5-bit.

It is very difficult to achieve such computation skipping in a software solution, since checking for zero-valued inputs in a scalar core consumes time. Using techniques for sparse matrix multiplication is not possible either, because the used convolutional kernels are typically very small on the order of 11×11 , 5×5 or 3×3 . However, computation skipping can be achieved in a hardware accelerator with dedicated hardware support. Flags can indicate if upcoming data is zero and prevent circuitry from switching if this is the case. As ConvNet-weights are fixed, these flags can be computed beforehand.

3.3. Achievable Energy-Accuracy Trade-Off in Convnets

By combining both precision scaling and algorithmical computation skipping, major energy savings can be achieved in ConvNets. Figure 6a shows the effect of uniform quantization on the energy consumption of our benchmark algorithms, if both precision scaling and computation skipping are combined. Note how the energy reduces 5 – 20× for an 8-bit implementation.

Figure 6b shows curves in the energy-accuracy space for our ConvNet benchmarks. It shows the trade-off for the 95-100% classification accuracy window. All curves are given for both uniform and per-layer quantization, and are compared to the typically used uniform 16-bit number representation, without adequate precision scaling and computation skipping. If per-layer quantization is used, the tradeoff is less steep than in the uniform case. This means more energy can be gained, while losing less classification accuracy in the process. All discussed networks gain at least an additional factor of 2 in energy consumption if a reduced classification accuracy of 99% is allowed.

Figure 7 gives an overview of the effect of each improvement on the relative energy consumption. Per-layer rescaling and computation skipping lead to the largest energy gains. Per-layer quantization leads to additional major energy gains if a reduced accuracy of 99% can be allowed.

SECTION 4. Conclusion

ConvNet algorithms are typically very computation and memory intensive. In order to be able to embed ConvNet-based classification into wearable platforms, their energy consumption should be reduced significantly. In this work, we show how the energy consumption of dedicated ConvNet accelerators can be drastically reduced by applying approximate computing. ConvNets for image classification are fault-tolerant and allow computation using 4–10 bits in stead of the previously used 16-or 32-bit formats. By using requantized number representations, the energy consumption in a ConvNet accelerator can be reduced in two complementary ways. First, energy consumption can be lower through precision scaling. Second, more quantization further increases the number of zeroed parameter and input values in the ConvNet. Since zero-valued numbers do not contribute to the algorithm's output, their computations can be skipped. The combination of these two techniques can lead to significant energy savings in several ConvNets. We show energy reductions of up to 30× compared to the commonly used 16-bit fixed point implementations, without sacrificing algorithm performance. If the classification accuracy can be reduced to 99%, additional energy savings can be achieved on the same network architecture, through per-layer optimization. In this case an additional 7.5× reduction can be achieved.

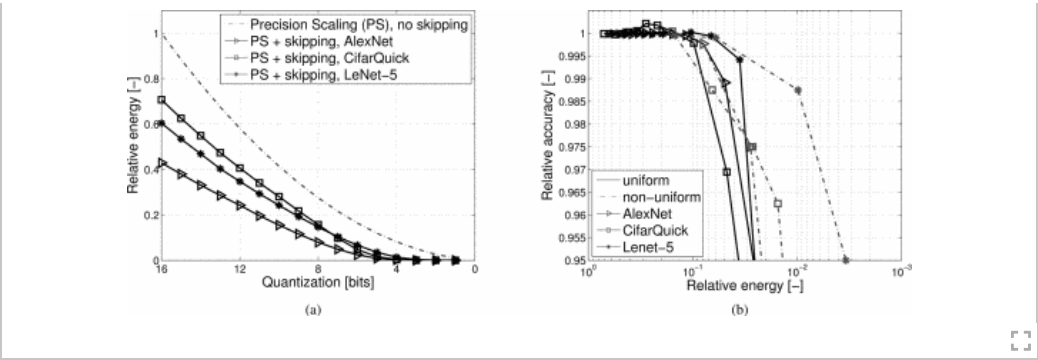


Figure 6: (a) Relative energy consumption for uniform quantization. A striking 5– 20 × lower energy consumption at 8-bit, compared to the commonly used 16-bit used in previous literature. (b) Energy-accuracy plot for per-layer quantization and scaling and with computation skipping. All considered convnet-networks gain at least an additional factor of 2 in energy consumption if 99% accuracy is allowed.

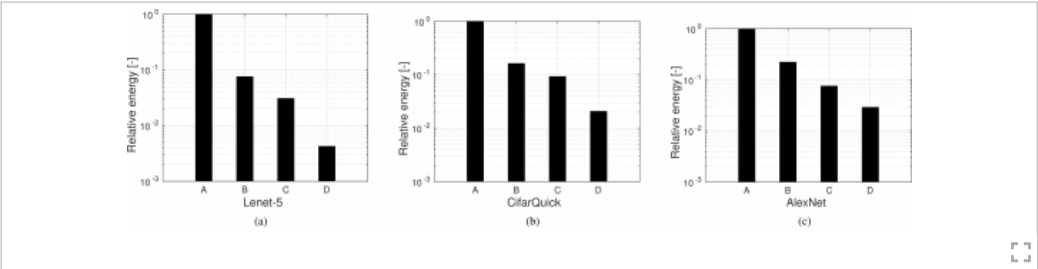


Figure 7: Energy comparison between our three benchmarks in different cases. (A) 16-bit (b) per-layer scaling, uniform quantization at 100% accuracy (c) same as B, with computation skipping (d) per-layer scaling, per-layer quantization and computation skipping at 99% accuracy.

Authors	▼
Figures	▼
References	▼
Citations	▼
Keywords	▼
Metrics	▼

[Back to Results](#)

IEEE Personal Account

CHANGE
USERNAME/PASSWORD

Purchase Details

PAYMENT OPTIONS
VIEW PURCHASED
DOCUMENTS

Profile Information

COMMUNICATIONS
PREFERENCES
PROFESSION AND
EDUCATION
TECHNICAL INTERESTS

Need Help?

US & CANADA: +1 800
678 4333
WORLDWIDE: +1 732
981 0060
CONTACT & SUPPORT

Follow

f @ in y

[About IEEE Xplore](#) | [Contact Us](#) | [Help](#) | [Accessibility](#) | [Terms of Use](#) | [Nondiscrimination Policy](#) | [IEEE Ethics Reporting](#)  | [Sitemap](#) | [IEEE Privacy Policy](#)

A public charity, IEEE is the world's largest technical professional organization dedicated to advancing technology for the benefit of humanity.

© Copyright 2025 IEEE - All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies.

EXHIBIT C

14.4 A 21.5M-Query-Vectors/s 3.37nJ/Vector Reconfigurable k-Nearest-Neighbor Accelerator with Adaptive Precision in 14nm Tri-Gate CMOS

Himanshu Kaul, Mark A. Anders, Sanu K. Mathew, Gregory Chen, Sudhir K. Satpathy, Steven K. Hsu, Amit Agarwal, Ram K. Krishnamurthy

Intel, Hillsboro, OR

Energy-efficient k-nearest-neighbor (kNN) computations are key building blocks for computer vision, classification, and machine-learning workloads [1-3]. Determining distances to high-dimensional vectors within a large vector database results in high compute cost. Adaptive precision improves energy efficiency by eliminating a majority of vectors without costly full-precision computation, with as-needed precision refinement to guarantee kNN accuracy of closely matched vectors. A special-purpose on-die kNN accelerator with 128-dimensions by 128 parallel reference vectors, targeted across mobile SoCs to multi-core microprocessors, and reconfigurable for either Manhattan or Euclidean distance, is fabricated in 14nm tri-gate CMOS [6]. Partial distance compute circuits, 2b window-based sort, MSB-to-LSB-based selective distance refinement, robust ultra-low voltage circuits, and state tracking control to selectively resume next-nearest candidates enable nominal energy efficiency of 3.37nJ/query vector or 9.7TOPS/W (measured for 21.5M vectors/s, 16 cycles/vector at 750mV, 25°C) with a dense layout occupying 0.333mm² (Fig. 14.4.7) while achieving: i) scalable performance up to 26.4M vectors/s, 114mW measured at 850mV, ii) 2-cycle latency and 43pJ energy to find each subsequent nearest neighbor, iii) up to 5.2× higher throughput while maintaining full-precision kNN accuracy, iv) 16× search-space reduction for next-nearest neighbor, v) ultra-low voltage operation measured at 360mV, 1.1M vectors/s, 1.44mW, and vi) peak energy efficiency of 1.23nJ/vector at 390mV (near-threshold), 25°C.

The kNN accelerator finds the nearest-neighbor indices using in-memory distance-compute circuits, a distributed sort network and efficient incremental search (Fig. 14.4.1). Parallel Manhattan and Euclidean distance computations to the 128 reference vectors (128-D per vector) start with low-accuracy from the MSB using partial distance compute circuits. The sort network finds the minimum within current accuracy bounds, eliminating vectors outside these bounds for up to 6× compute reduction. The partial distance compute circuits refine the remaining distances, with a successive 2b of the accumulated distance available for further sorting in each iteration. Search-space control circuits coordinate these partial distance iterations, vector eliminations and 2b sorting windows until the nearest neighbor is found. Since the last eliminated group of vectors will contain the next nearest neighbor, tracking these groups enables an efficient incremental search within only the last group. Early vector elimination improves throughput up to 5.2×, and narrow datapath width reduces accelerator area by 46%, while maintaining full-precision kNN accuracy.

Complete Euclidean and Manhattan distances from dimension-wise 8b absolute differences require accumulation over 6 and 4 partial-compute iterations, respectively (Fig. 14.4.2a). Iterative refinement of accumulated distance is interleaved as needed with successive sort operations on 2b windows from MSB to LSB. Area-optimized absolute difference uses multi-bit latches for reference vector storage and ripple-carry static mirror adders (Fig. 14.4.2b). The MSB carry-out connects to the LSB carry-in for 2's complement absolute value increment, replacing a separate incrementer for 30% lower area. Use of partial-squaring operations reduce each 8b multiplier to a pair of 2b multipliers, with the shuffle circuit selecting the appropriate inputs from the absolute difference. Narrow partial distance bit-widths also reduce compressor tree area and energy. The 4b products from each dimension are added separately to relocate the left-shift and zero-select circuits after the compressor tree, reducing them from one per dimension to one per vector. The shared partial-squaring and compressor-tree circuits are multiplexed between every pair of vectors using the *vector select* signal for 28% lower area. The previously accumulated distance is shifted left 2b in every compute iteration, enabling replacement of an 11b right shift in the critical path for incremental distance alignment with only a 1b right shift (Fig. 14.4.3). This also decreases the radix-2 carry-merge adder width to 12b for incremental distance summation, with a non-critical 10b incrementer for the upper bits. These optimizations reduce accumulator delay by 37% and area by 48%. Each distance refinement can affect previously sorted 2b windows by at most 1, which must be included in each partial-sort iteration. Therefore, the 12:1 multiplexers forward a

carry-out or prior difference (Y) along with the correct accumulator 2b window to the sort network. The *precise* signals indicate whether all other *psums* exceed the minimum by more than 1, signifying that the same minimum has been found as with full-precision calculations. Decreasing comparator width from 23b to 3b simultaneously reduces sort network area by 3.6× and latency by 2×. Overall energy to find the nearest neighbor reduces up to 3× compared to a full-precision design.

Search-space control enables tracking: i) the vectors that could be possible winners until only one remains, ii) the current 2b window indicated by a global pointer, and iii) 2b windows with eliminated vectors for the next nearest neighbor search. The control-circuit location is divided among vector-compute blocks and a centralized global control (Fig. 14.4.4). The *compare psum* circuit uses the global minimum to determine vector elimination and nearest neighbor selection. Early removal of vectors while moving MSB to LSB for nearest-neighbor search results in 47% compute reduction and a 2b distance sort reduction of 3.8×. The global binary mask stores a one when vectors are eliminated as the global pointer advances MSB to LSB. The global pointer rolls back to the closest 2b window in this binary mask since the vectors removed in this window contain the next-nearest neighbor. Each vector stores the pointer position at elimination, and during rollback the *compare pointer* circuits re-enable only the closest vectors. This search for the next nearest neighbor within a selective subset of the remaining vectors results in 16× search-space reduction.

The adaptive precision kNN accelerator finds the nearest neighbor out of 128×128-D randomly generated vectors for 21.5M query vectors/s at 16 cycles/vector with Manhattan distance measured at 338MHz, nominal 750mV, 25°C (Fig. 14.4.5). Corresponding total power consumption is 73mW (1.8mW active leakage component), resulting in energy-per-query vector of 3.37nJ or 9.7TOPS/W (10P = 8b add/multiply/compare). As k scales from 1 to 10, mean increase in latency and energy per k is 2 cycles and 43pJ. Nominal Euclidean operation is 20.3M vectors/s at 17 cycles/vector, consuming 78mW or 3.84nJ/vector. Mean increase in latency and energy per k is 4 cycles and 87pJ. Peak Manhattan (Euclidean) performance scales up to 26.4M (24.8M) vectors/s with power consumption of 114mW (123mW) at 850mV (Fig. 14.4.6). Datapath circuits optimized for reliable ultra-low-voltage operation [4] enable robust functionality measured down to 360mV, with k=1 throughput scaling to 1.1M vectors/s, 1.44mW for Manhattan and 1.04M vectors/s, 1.63mW for Euclidean computations. Energy decreases with supply voltage for a peak efficiency of 1.23nJ per query vector or 26.5TOPS/W with Manhattan (k=1) mode at 390mV (near-threshold), representing a 2.73× energy efficiency improvement over nominal 750mV operation. Energy-efficiency measurements also improve with algorithm-dependent SURF-128 [5] scale- and rotation-invariant feature vector matching for object recognition. These feature vectors achieve an additional 15% (Manhattan) and 20% (Euclidean) energy savings at 750mV, resulting in 2.86nJ/vector and 3.07nJ/vector, respectively.

Acknowledgments:

The authors thank M. Haycock, M. Mayberry, H. Jiang, S. Borkar, V. De, M. Kounavis and F. Ghenassia for encouragement and discussions. This research was, in part, funded by the U.S. Government. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government.

References:

- [1] Y. Kim et al., "A 0.5V 54μW Ultra-Low-Power Recognition Processor with 93.5% Accuracy Geometric Vocabulary Tree and 47.5% Database Compression," *ISSCC Dig. Tech. Papers*, pp. 330-331, 2015.
- [2] I. Hong et al., "A 125,582 vector/s Throughput and 95.1% Accuracy ANN Searching Processor with Neuro-Fuzzy Vision Cache for Real-time Object Recognition," *IEEE Symp. on VLSI Circuits*, pp. 184-185, 2013.
- [3] T.-W. Chen et al., "A Multimedia Semantic Analysis SoC (SASoC) with Machine-Learning Engine," *ISSCC Dig. Tech. Papers*, pp. 338-339, 2010.
- [4] S. Hsu et al., "A 280mV-to-1.1V 256b Reconfigurable SIMD Vector Permutation Engine with 2-Dimensional Shuffle in 22nm CMOS," *ISSCC Dig. Tech. Papers*, pp. 178-179, 2012.
- [5] H. Bay et al., "SURF: Speeded Up Robust Features," *J. Computer Vision and Image Understanding*, 110(3):346-359, 2008.
- [6] C.-H. Jan et al., "A 14 nm SoC Platform Technology Featuring 2nd Generation Tri-Gate Transistors, 70 nm Gate Pitch, 52 nm Metal Pitch, and 0.0499 μm² SRAM Cells, Optimized for Low Power, High Performance and High Density SoC Products," *IEEE Symp. on VLSI Tech.*, pp. 12-13, 2015.

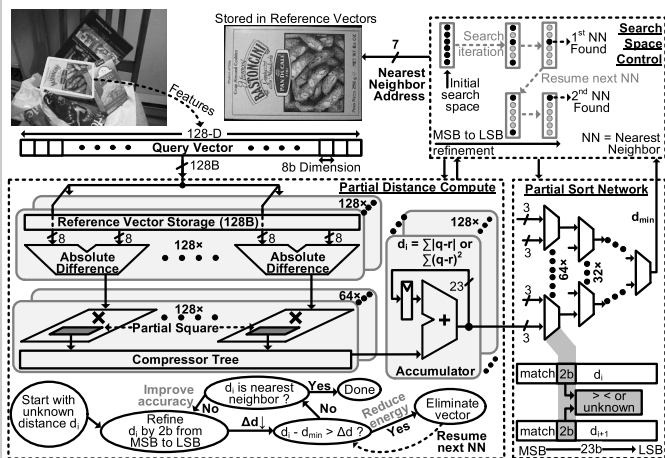


Figure 14.4.1: kNN accelerator organization.

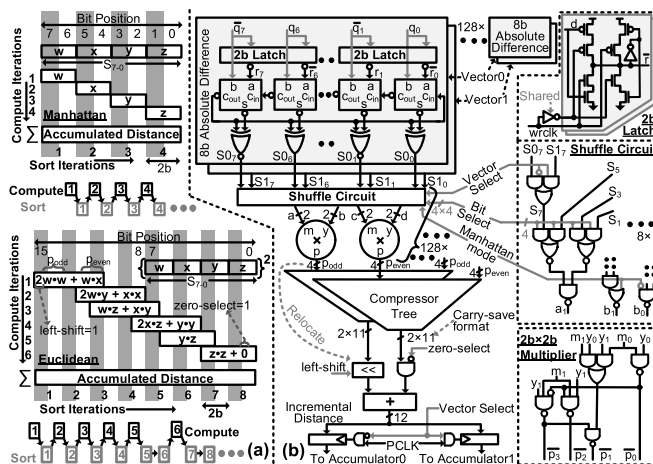


Figure 14.4.2: Partial distance compute (a) iterations and (b) circuits.

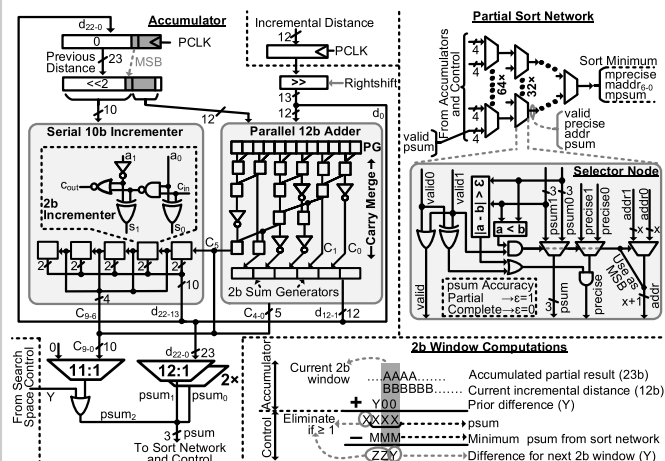


Figure 14.4.3: Accumulator and sort circuits.

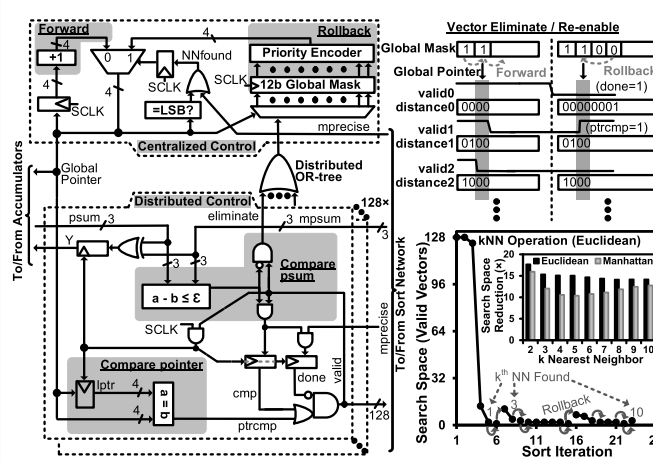


Figure 14.4.4: Search space control circuits.

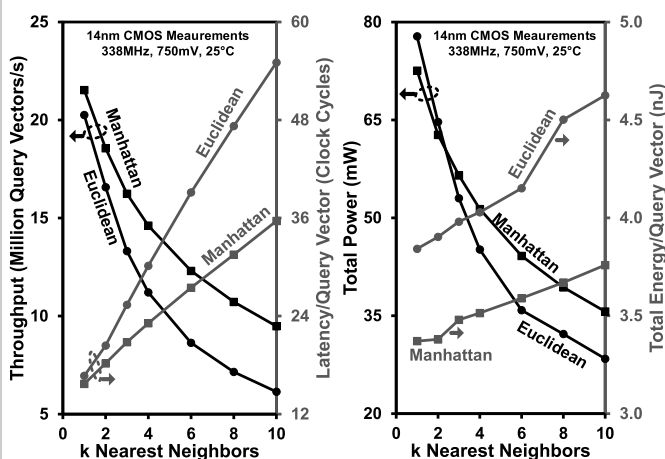


Figure 14.4.5: 14nm CMOS measured throughput, latency, total power and energy vs. number of nearest neighbors computed (k).

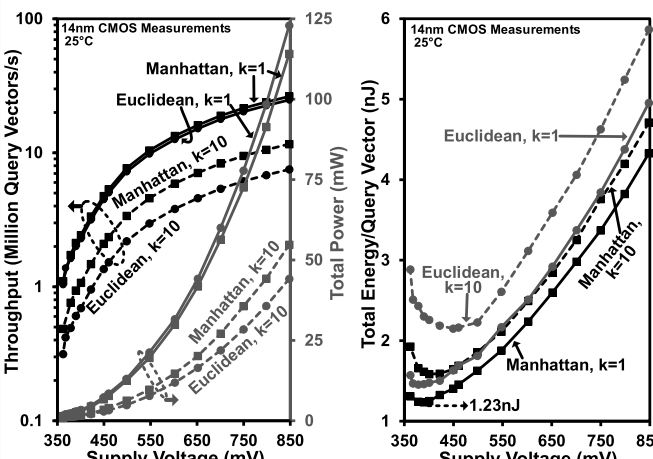
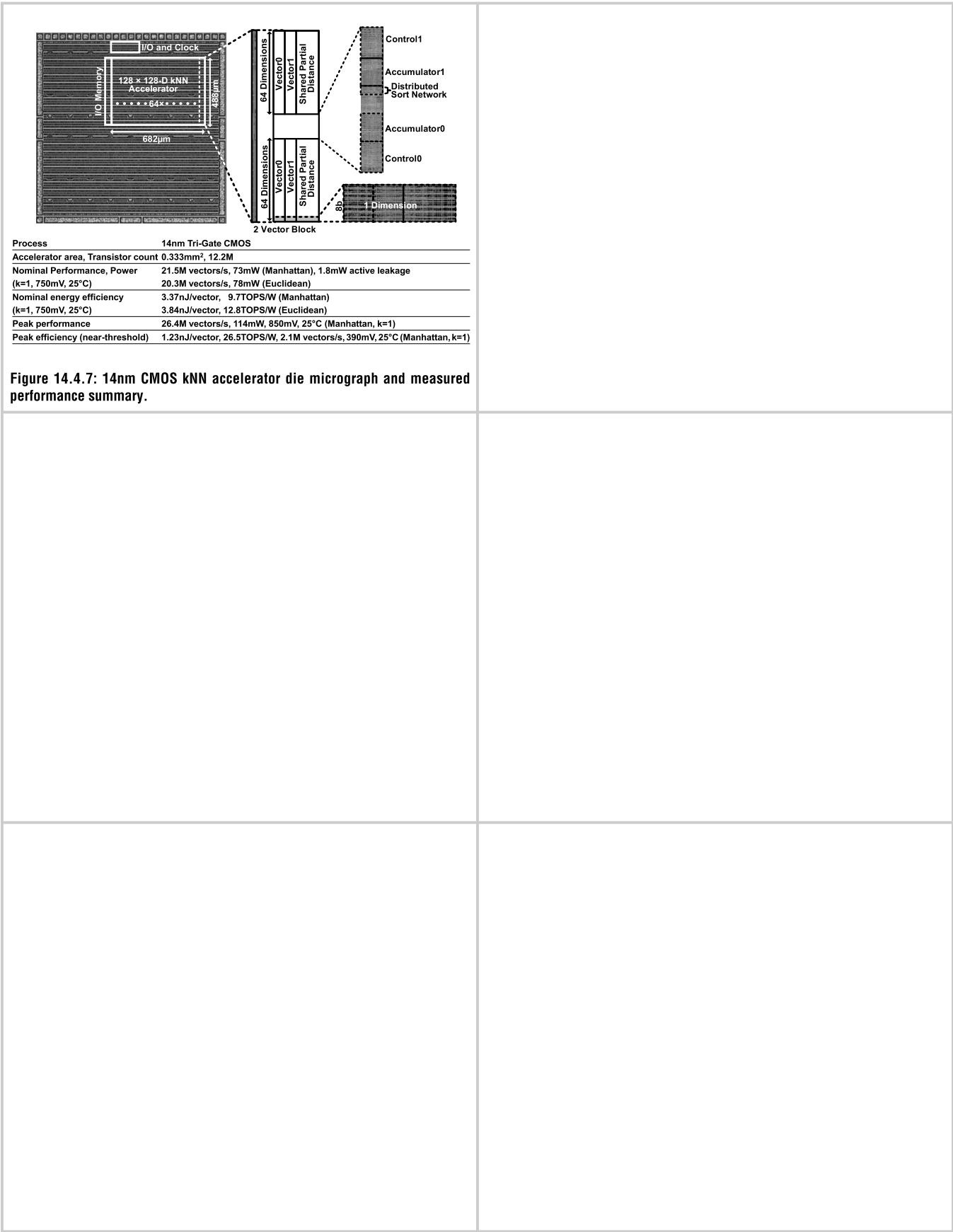


Figure 14.4.6: 14nm CMOS measured throughput, total power and energy vs. supply voltage.



ABSTRACT C

14.4 A 21.5M-query-vectors/s 3.37nJ/vector reconfigurable k-nearest-neighbor accelerator with adaptive precision in 14nm tri-gate CMOS

Publisher: IEEE Cite This PDF

Himanshu Kaul ; Mark A. Anders ; Sanu K. Mathew ; Gregory Chen ; Sudhir K. Satpathy ; Steven K. Hsu All Authors

7 Cites in Papers 1 Cites in Patent 1178 Full Text Views

Abstract	Abstract: Energy-efficient k-nearest-neighbor (kNN) computations are key building blocks for computer vision, classification, and machine-learning workloads [1-3]. Determining distances to high-dimensional vectors within a large vector database results in high compute cost. Adaptive precision improves energy efficiency by eliminating a majority of vectors without costly full-precision computation, with as-needed precision refinement to guarantee kNN accuracy of closely matched vectors. A special-purpose on-die kNN accelerator with 128-dimensions by 128 parallel reference vectors, targeted across mobile SoCs to multi-core microprocessors, and reconfigurable for either Manhattan or Euclidean distance, is fabricated in 14nm tri-gate CMOS [6]. Partial distance compute circuits, 2b window-based sort, MSB-to-LSB-based selective distance refinement, robust ultra-low voltage circuits, and state tracking control to selectively resume next-nearest candidates enable nominal energy efficiency of 3.37nJ/query vector or 9.7TOPS/W (measured for 21.5M vectors/s, 16 cycles/vector at 750mV, 25°C) with a dense layout occupying 0.333mm2 (Fig. 14.4.7) while achieving: i) scalable performance up to 26.4M vectors/s, 114mW measured at 850mV, ii) 2-cycle latency and 43pJ energy to find each subsequent nearest neighbor, iii) up to 5.2× higher throughput while maintaining full-precision kNN accuracy, iv) 16× search-space reduction for next-nearest neighbor, v) ultra-low voltage operation measured at 360mV, 1.1M vectors/s, 1.44mW, and vi) peak energy efficiency of 1.23nJ/vector at 390mV (near-threshold), 25°C.
Authors	
Figures	
References	
Citations	
Keywords	
Metrics	
More Like This	

Published in: 2016 IEEE International Solid-State Circuits Conference (ISSCC)
Date of Conference: 31 January 2016 - 04 February 2016 **DOI:** 10.1109/ISSCC.2016.7418006
Date Added to IEEE Xplore: 25 February 2016 **Publisher:** IEEE
► ISBN Information: **Conference Location:** San Francisco, CA, USA
Electronic ISSN: 2376-8606

Energy-efficient k-nearest-neighbor (kNN) computations are key building blocks for computer vision, classification, and machine-learning workloads [1]– [3]. Determining distances to high-dimensional vectors within a large vector database results in high compute cost. Adaptive precision improves energy efficiency by eliminating a majority of vectors without costly full-precision computation, with as-needed precision refinement to guarantee kNN accuracy of closely matched vectors. A special-purpose on-die kNN accelerator with 128-dimensions by 128 parallel reference vectors, targeted across mobile SoCs to multi-core microprocessors, and reconfigurable for either Manhattan or Euclidean distance, is fabricated in 14nm tri-gate CMOS [6]. Partial distance compute circuits, 2b window-based sort, MSB-to-LSB-based selective distance refinement, robust ultra-low voltage circuits, and state tracking control to selectively resume next-nearest candidates enable nominal energy efficiency of 3.37nJ/query vector or 9.7TOPS/W (measured for 21.5M vectors/s, 16 cycles/vector at 750mV, 25°C) with a dense layout occupying 0.333mm² (Fig. 14.4.7) while achieving: i) scalable performance up to 26.4M vectors/s, 114mW measured at 850mV, ii) 2-cycle latency and 43pJ energy to find each subsequent nearest neighbor, iii) up to 5.2× higher throughput while maintaining full-precision kNN accuracy, iv) 16× search-space reduction for next-nearest neighbor, v) ultra-low voltage operation measured at 360mV, 1.1 M vectors/s, 1.44mW, and vi) peak energy efficiency of 1.23nJ/Vector at 390mV (near-threshold), 25°C.

The kNN accelerator finds the nearest-neighbor indices using in-memory distance-compute circuits, a distributed sort network and efficient incremental search (Fig. 14.4.1). Parallel Manhattan and Euclidean distance computations to the 128 reference vectors (128-D per vector) start with low-accuracy from the MSB using partial distance compute circuits. The sort network finds the minimum within current accuracy bounds, eliminating vectors outside these bounds for up to 6 $\times$ compute reduction. The partial distance compute circuits refine the remaining distances, with a successive 2b of the accumulated distance available for further sorting in each iteration. Search-space control circuits coordinate these partial distance iterations, vector eliminations and 2b sorting windows until the nearest neighbor is found. Since the last eliminated group of vectors will contain the next nearest neighbor, tracking these groups enables an efficient incremental search within only the last group. Early vector elimination improves throughput up to 5.2 $\times$, and narrow datapath width reduces accelerator area by 46%, while maintaining full-precision kNN accuracy.

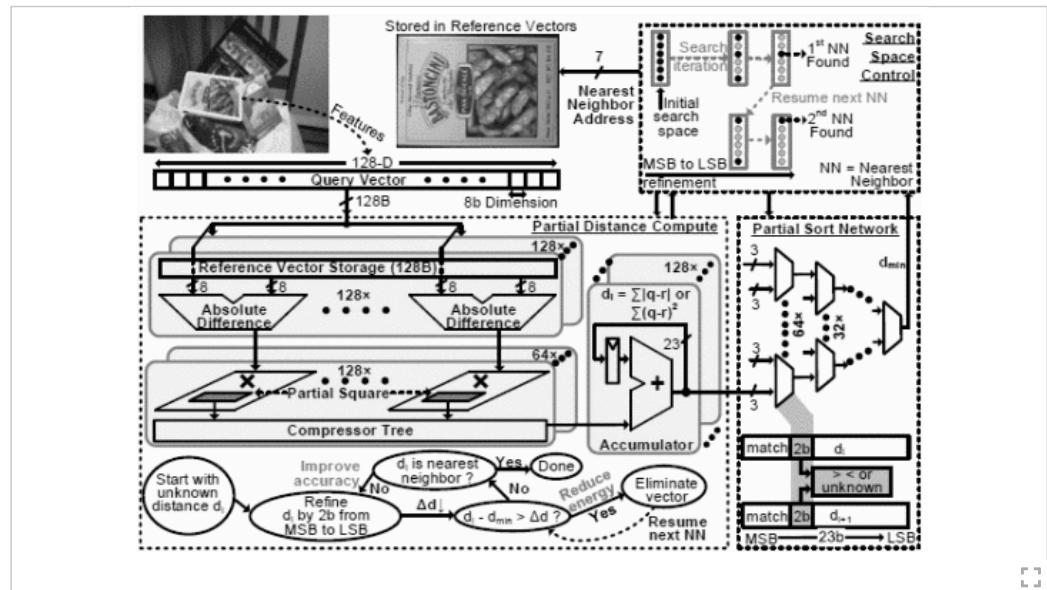


Figure 14.4.1:
kNN accelerator organization.

Complete Euclidean and Manhattan distances from dimension-wise 8b absolute differences require accumulation over 6 and 4 partial-compute iterations, respectively (Fig. 14.4.2a). Iterative refinement of accumulated distance is interleaved as needed with successive sort operations on 2b windows from MSB to LSB. Area-optimized absolute difference uses multi-bit latches for reference vector storage and ripple-carry static mirror adders (Fig. 14.4.2b). The MSB carryout connects to the LSB carry-in for 2's complement absolute value increment, replacing a separate incrementer for 30% lower area. Use of partial-squaring operations reduce each 8b multiplier to a pair of 2b multipliers, with the shuffle circuit selecting the appropriate inputs from the absolute difference. Narrow partial distance bit-widths also reduce compressor tree area and energy. The 4b products from each dimension are added separately to relocate the left-shift and zero-select circuits after the compressor tree, reducing them from one per dimension to one per vector. The shared partial-squaring and compressor-tree circuits are multiplexed between every pair of vectors using the *vector select* signal for 28% lower area. The previously accumulated distance is shifted left 2b in every compute iteration, enabling replacement of an 11b right shift in the critical path for incremental distance alignment with only a 1b right shift (Fig. 14.4.3). This also decreases the radix-2 carry-merge adder width to 12b for incremental distance summation, with a non-critical 10b incrementer for the upper bits. These optimizations reduce accumulator delay by 37% and area by 48%. Each distance refinement can affect previously sorted 2b windows by at most 1, which must be included in each partial-sort iteration. Therefore, the 12:1 multiplexers forward a carry-out or prior difference (*Y*) along with the correct accumulator 2b window to the sort network. The *precise* signals indicate whether all other *psums* exceed the minimum by more than 1, signifying that the same minimum has been found as with full-precision calculations. Decreasing comparator width from 23b to 3b simultaneously reduces sort network area by 3.6 $\times$ and latency by 2 $\times$. Overall energy to find the nearest neighbor reduces up to 3 $\times$ compared to a full-precision design.

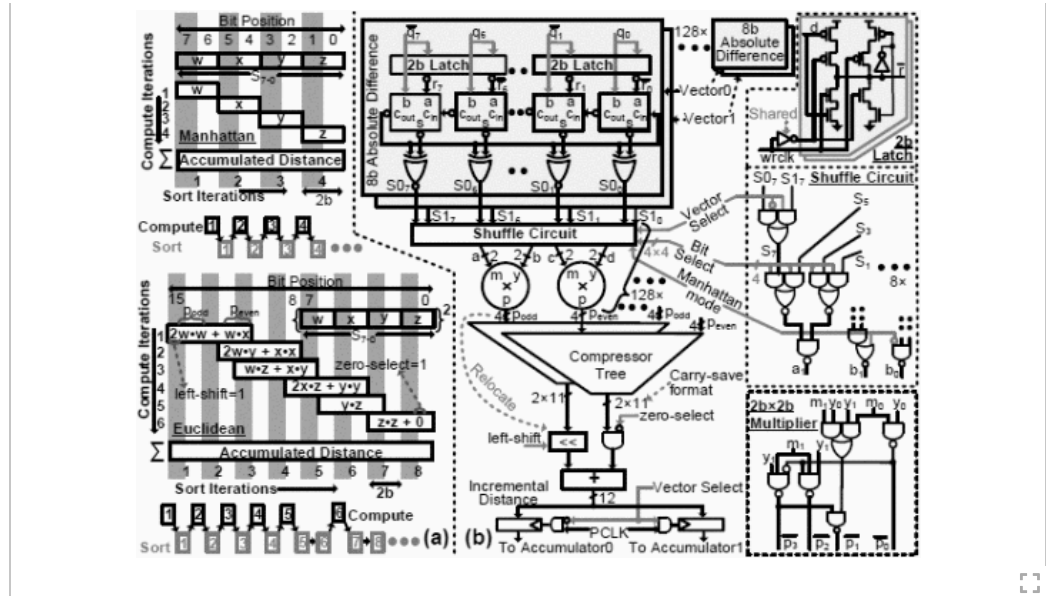


Figure 14.4.2: Partial distance compute (a) iterations and (b) circuits.

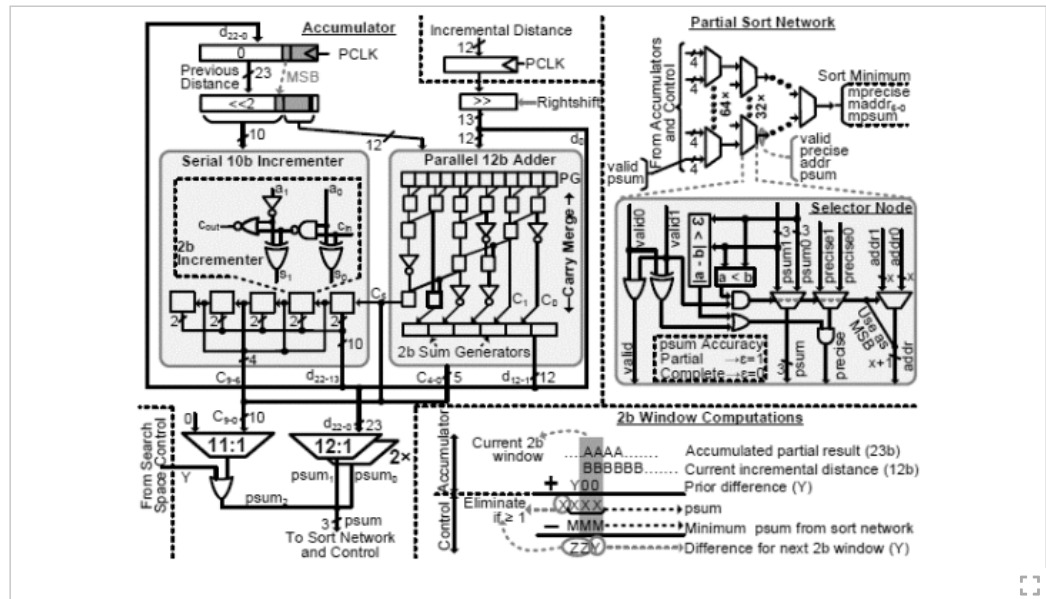


Figure 14.4.3: Accumulator and sort circuits.

Search-space control enables tracking: i) the vectors that could be possible winners until only one remains, ii) the current 2b window indicated by a global pointer, and iii) 2b windows with eliminated vectors for the next nearest neighbor search. The control-circuit location is divided among vector-compute blocks and a centralized global control (Fig. 14.4.4). The *compare psum* circuit uses the global minimum to determine vector elimination and nearest neighbor selection. Early removal of vectors while moving MSB to LSB for nearest-neighbor search results in 47% compute reduction and a 2b distance sort reduction of 3.8×. The global binary mask stores a one when vectors are eliminated as the global pointer advances MSB to LSB. The global pointer rolls back to the closest 2b window in this binary mask since the vectors removed in this window contain the next-nearest neighbor. Each vector stores the pointer position at elimination, and during rollback the *compare pointer* circuits re-enable only the closest vectors. This search for the next nearest neighbor within a selective subset of the remaining vectors results in 16× search-space reduction.

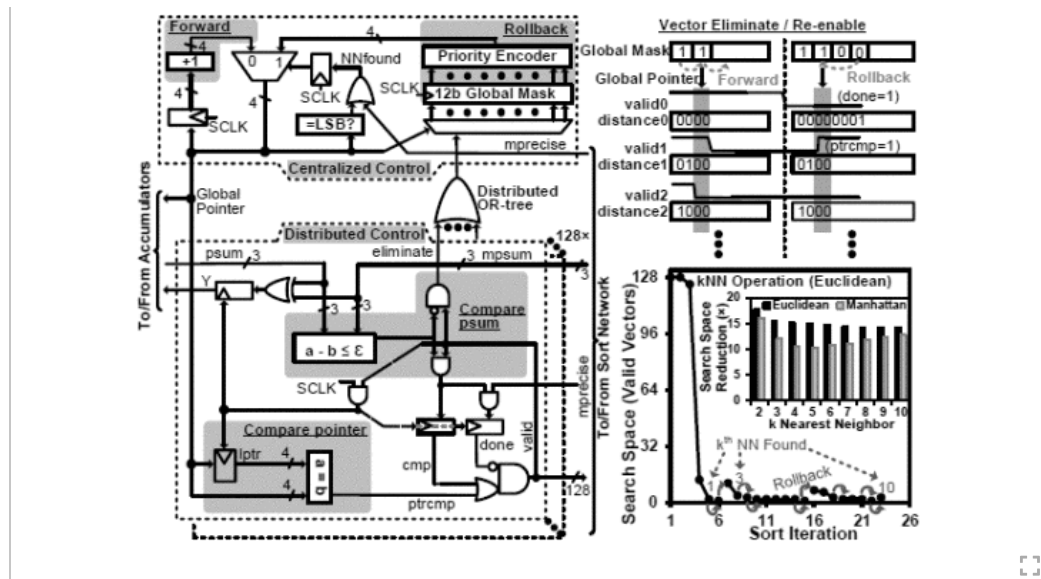


Figure 14.4.4:
Search space control circuits.

The adaptive precision kNN accelerator finds the nearest neighbor out of 128×128 -D randomly generated vectors for 21.5M query vectors/s at 16 cycles/vector with Manhattan distance measured at 338MHz, nominal 750mV, 25°C (Fig. 14.4.5). Corresponding total power consumption is 73mW (1.8mW active leakage component), resulting in energy-per-query vector of 3.37nJ or 9.7TOPS/W (1OP = 8b add/multiply/compare). As k scales from 1 to 10, mean increase in latency and energy per k is 2 cycles and 43pJ. Nominal Euclidean operation is 20.3M vectors/s at 17 cycles/vector, consuming 78mW or 3.84nJ/vector. Mean increase in latency and energy per k is 4 cycles and 87pJ. Peak Manhattan (Euclidean) performance scales up to 26.4M (24.8M) vectors/s with power consumption of 114mW (123mW) at 850mV (Fig. 14.4.6). Datapath circuits optimized for reliable ultra-low-voltage operation [4] enable robust functionality measured down to 360mV, with k=1 throughput scaling to 1.1 M vectors/s, 1.44mW for Manhattan and 1.04M vectors/s, 1.63mW for Euclidean computations. Energy decreases with supply voltage for a peak efficiency of 1.23nJ per query vector or 26.5TOPS/W with Manhattan (k=1) mode at 390mV (near-threshold), representing a $2.73 \times$ energy efficiency improvement over nominal 750mV operation. Energy-efficiency measurements also improve with algorithm-dependent SURF-128 [5] scale- and rotation-invariant feature vector matching for object recognition. These feature vectors achieve an additional 15% (Manhattan) and 20% (Euclidean) energy savings at 750mV, resulting in 2.86nJ/vector and 3.07nJ/vector, respectively.

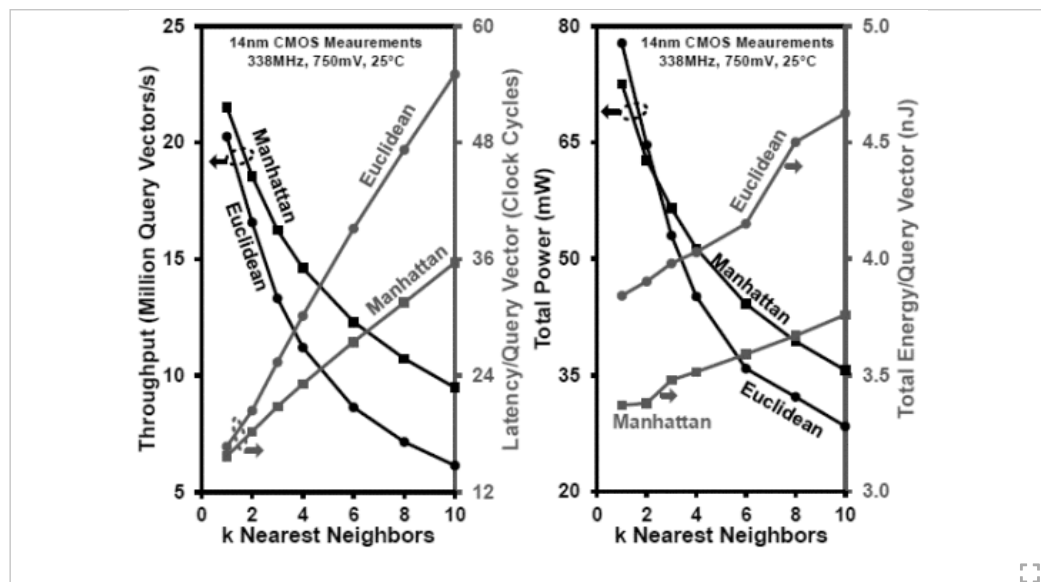


Figure 14.4.5:
14nm CMOS measured throughput, latency, total power and energy vs. number of nearest neighbors computed (k).

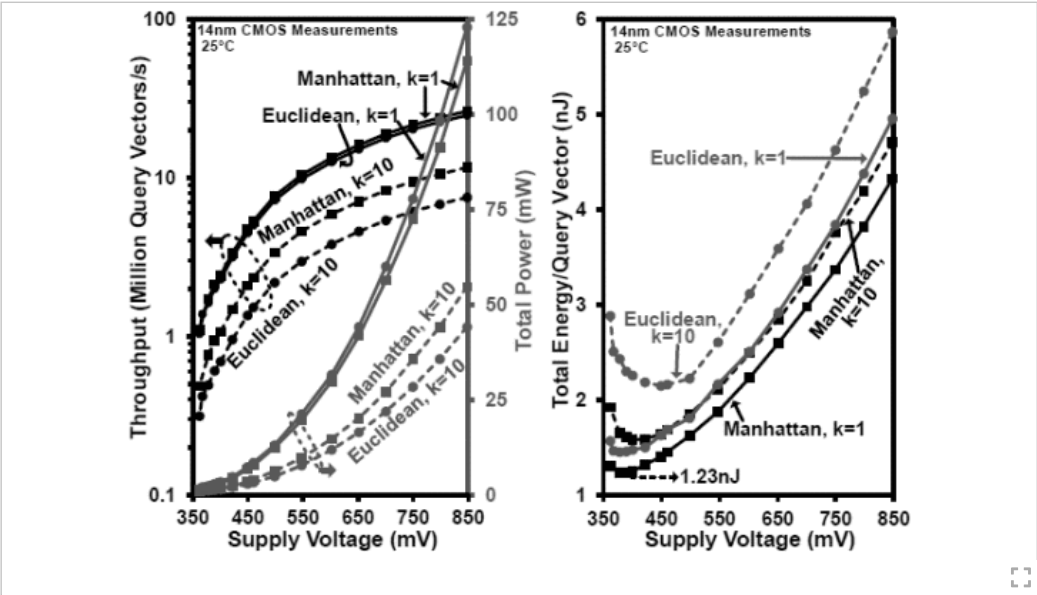


Figure 14.4.6:
14nm CMOS measured throughput, total power and energy vs. Supply voltage.

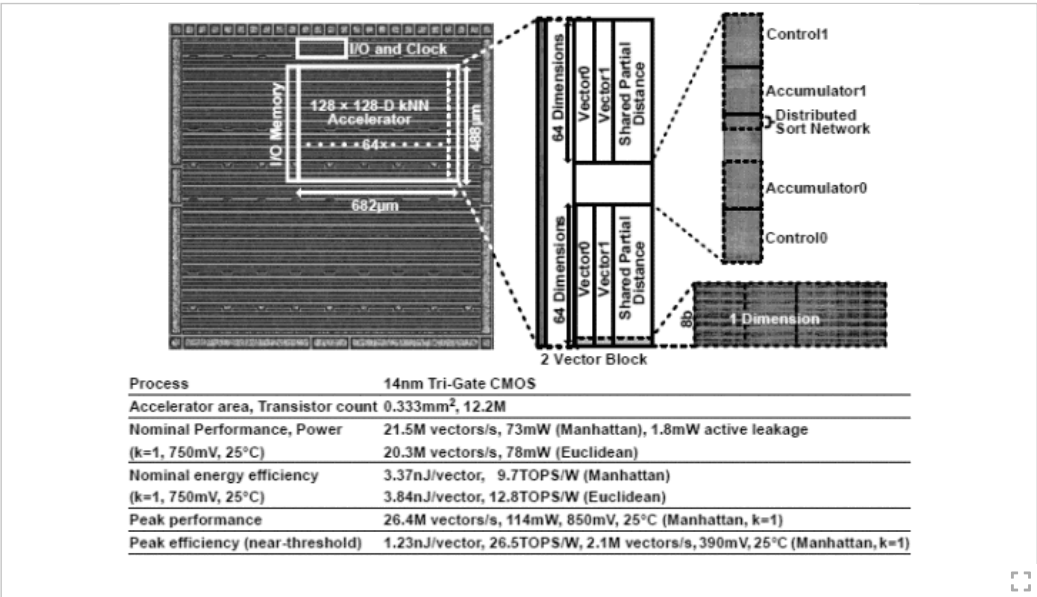


Figure 14.4.7:
14nm CMOS kNN accelerator die micrograph and measured performance summary.

ACKNOWLEDGMENTS

The authors thank M. Haycock, M. Mayberry, H. Jiang, S. Borkar, V. De, M. Kounavis and F. Ghenassia for encouragement and discussions. This research was, in part, funded by the U.S. Government. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government.

Authors	▼
Figures	▼
References	▼

Citations	▼
Keywords	▼
Metrics	▼

Back to Results

IEEE Personal Account	Purchase Details	Profile Information	Need Help?	Follow
CHANGE USERNAME/PASSWORD	PAYMENT OPTIONS VIEW PURCHASED DOCUMENTS	COMMUNICATIONS PREFERENCES PROFESSION AND EDUCATION TECHNICAL INTERESTS	US & CANADA: +1 800 678 4333 WORLDWIDE: +1 732 981 0060 CONTACT & SUPPORT	f @ in v