Detecting and Bypassing Trivial Computations in Convolutional Neural Networks

Dongning Ma, Xun Jiao
Department of Electrical and Computer Engineering, Villanova University
{dma2,xjiao}@villanova.edu

***Abstract*—Convolutional neural networks (CNNs) recently are able to exceed human accuracy in various application domains such as image recognition, medical diagnosis, and financial analysis. However, the high computational complexity of CNNs incurs high energy consumption on current hardware implementations. Existing solutions such as pruning and quantization typically require retraining or fine-tuning to regain accuracy, which can be cost-prohibitive and time-consuming. This paper proposes a retraining-free approach to reducing the computation workload of CNNs during inference by detecting and bypassing the trivial computations. We define trivial computations as the computations the results of which can be determined without actual computations. The examples include multiplication with 0, +1/-1 and addition with 0 or addition of opposite numbers. Correspondingly, we develop bypass circuits to detect the trivial computations. Once detected, the circuit delivers the pre-determined result without an actual computation. Experimental results on MNIST and EMNIST show that the CNNs with bypass circuits can lead to 30.66-33.52% energy savings without any accuracy loss. This technique can be used together with existing techniques such as pruning and quantization because it is totally complimentary to such techniques.**

I. Introduction

Convolutional neural networks (CNNs) have achieved tremendous success on many application domains such as image classification [16], medical diagnosis [6], and speech recognition [12]. This versatility has led their implementations on various hardware platforms. However, the high accuracy of CNNs comes at the cost of high computation workload. Even for a single input query, CNNs require billions of addition and multiplication operations [3], incuring high energy consumption. This remains a challenge for their deployment in hardware.

To combat this challenge, recent approaches reduce computing workloads by exploring the use of approximations in computed results, often referred to as "Approximate Computing." Approximate computing leverages error tolerance at the application level to allow inaccurate computations in a system as long as the output quality is acceptable. Neural networks are naturally suited to such approximations because of their intrinsic error tolerance [5], [20], [23]. Existing works leverage pruning-based techniques to shrink the original dense network size, which leads to a sparse network structure [9], [10]. Another set of works use quantization that converts the floating point operations into fixed point operations [19], [8]. Selective replacement of the less-critical neurons with approximate neurons is proposed in [23]. The substitution of exact multipliers with approximate multipliers is proposed in [5], [20]. While the adaptability of neural networks makes it a natural target for approximation, in practice it also requires *retraining* or fine-tuning to mitigate approximation-induced errors, which can be cost-prohibitive and time-consuming.

To overcome above-mentioned limitations, we propose a retraining-free method to reduce the computation workload of CNNs by detecting and bypassing the trivial computations. This method is motivated by the strong data locality in CNNs as shown in Fig. 4, where we present the addition and multiplication operands distribution in a CNN. Based on the distribution, we observe that many computation results can be determined in advance without actual computations such as multiplication with 0, +1/-1 and addition with 0. We define these computations as *trivial computations*. Correspondingly, we then design bypass circuits that can detect such trivial computations. Once detected, the bypass circuits will return the pre-determined results.

The advantages of this method are four-fold: retraining-free, no accuracy drop, low overhead, and totally complimentary to existing methods such as pruning and quantization. Thus, they can be used together if necessary. The bypass circuits are tightly integrated with computation units in hardware to detect and bypass the computations based on the incoming input operands. With a small area penalty (around 3%), the bypass circuit can be applied to almost any hardware platforms such as GPU, FPGA, and ASIC.

Our contributions are as follows:

- At the software level, we explore the data locality in CNNs, based on which we define and classify trivial computations into four cases. We quantify the portion of each case existing in CNNs.
- At the hardware level, we implement bypass circuits that will detect and bypass trivial computations. We design and physically implement gate-level bypass circuits to measure their energy/area overhead.
- We evaluate the effectiveness of the proposed approach on MNIST and EMNIST. Experimental results show that with zero accuracy loss and a small area penalty (around 3%), our approach results in 30.66-33.52% energy saving over baseline architecture without computation bypass.

The remainder of this paper is organized as follows: Section II describes the necessary background on CNNs. Section III describes the proposed approach. Section IV presents the experimental results, while Section V discusses some related work. Section VI concludes the paper.

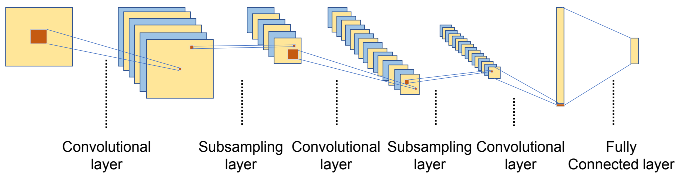


Fig. 1. An illustration of a convolutional neural network.

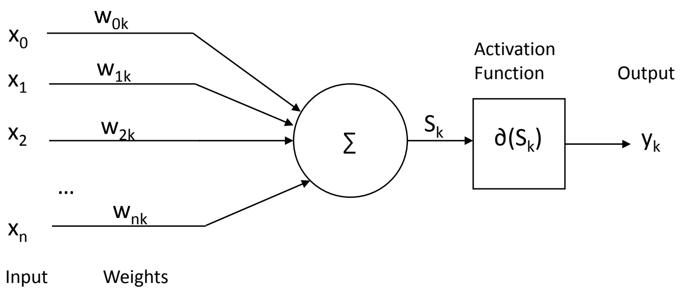


Fig. 2. The computation processes of an artificial neuron.

II. Background

Modeled after brain-inspired biological neuronal processing, (artificial) neural networks are a family of problem-solving methods in machine learning. Recently, convolutional neural networks (CNNs) have been increasingly popular in various applications due to their superior accuracy [12]. As shown in Fig. 1, a CNN typically consists of an input layer, several hidden layers, and an output layer. The hidden layers include convolutional layers, subsampling layers, and fully connected layers. Fig. 1 depicts LeNet-5 [18], a typical CNN architecture that consists of six layers, where the first, third, and fifth layer are convolutional, while the second, fourth are pooling layers, and the sixth layer is a fully connected layer.

Convolutional layers contain a number of kernels, each of which consists of several filters. The convolution operation models the hardwired bonding between the neurons. Sliding filters are used to perform dot-products of the filter and uses a portion of the input image to generate an output image, namely the feature map. The basic units in a neural network are called artificial neurons that perform the basic computations illustrated in Fig. 2. The computation process of a typical neuron consists of linear processing followed by non-linear processing. The linear processing requires the inputs to multiply the corresponding weights and all products are then accumulated. Later in the non-linear processing, the accumulation result is fed into an activation function such as rectilinear unit (ReLU) to be mapped to a specific range. Finally, the output y_k of neuron k is computed as $y_k = \delta(\sum_{j=1}^{n} x_j w_{jk} - \theta)$, where x_j is the j^{th} input, w_{jk} is the synaptic weight connecting j^{th} input and neuron k, θ is the bias, and δ is the activation function. While fixed point implementations of neural networks are widely used in mobile platforms, floating point implementations are still preferred in data centers because they prefer accurate models like ResNet [11] and ResNeXt [24]. In this paper, we target CNNs with floating point operations. These floating point operations are completed by floating point units (FPUs).

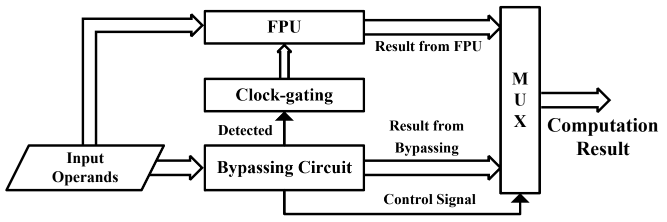


Fig. 3. Overview of trivial computations bypass.

III. Proposed Approach

A. Approach Overview

Fig. 3 presents an overview of the proposed bypass circuits. It is comprised of the following steps:

- For each computation, the two input operands are sent to both FPU and the bypass circuit.
- The bypass circuit will detect whether it is a trivial computation according to Table I. If a trivial computation is detected, the bypass circuit will "clock-gate" the FPU to stop its ongoing computation.
- The bypass circuit will then generate and send a control signal to the multiplexer according to what type the computation belongs to.
- The multiplexer will select and output the computation result based on the control signal. If there is no trivial computation detected, the multiplexer will output the computation result from FPU.

The rest of this section describes how we define and classify the trivial computations, based on which we develop bypass circuits to detect and bypass them.

B. Trivial Computations in CNNs

Classification Trivial computations are specific computations of which results can be determined without actual computations. Specifically, in the context of CNNs where we mainly focus on additions and multiplications, we classify the computations into the following cases as illustrated in Table I. We classify the trivial computation in multiplications ((a), (b)) and additions ((d) and (e)). We bypass the trivial computations according to the result section in Table I to avoid actual floating point operations.

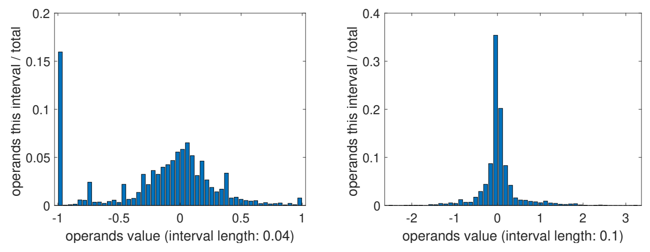
In fact, case (a) represents multiplications with one operand being zero, which is defined as *ineffectual multiplications* in existing works [2], [15], [14]. In fact, we notice and point out in Section IV that case (a) only represents a small portion of the overall trivial computations. By exploiting other cases (case (b)/(d)/(e)), we are able to better leverage the intrinsic properties of computation in CNNs.

Distribution To explore the distribution of trivial computations in CNNs, we first draw the histogram of MUL and ADD operands from the convolutional layers in which most computations of CNNs are completed. As shown in Fig. 4, operands

TABLE I
CLASSIFICATION OF COMPUTATIONS INSIDE CNNS.

computation		description		result
MUL	(a)	any of the operands equal to 0	$(a = 0 \text{ or } b = 0)$	0
	(b)	any of the operands equal to 1 or -1	$(\|a\| = 1 \text{ or } \|b\| = 1)$	$\pm b$ or $\pm a$
	(c)	the operands belong to neither of the cases above		$a \times b$
ADD	(d)	any of the operands equals to 0	$(a = 0 \text{ or } b = 0)$	b or a
	(e)	the two operands are inverse numbers of each other	$(a + b = 0)$	0
	(f)	the operands belong to neither of the cases above		$a + b$

a and b refer to the two operands involved in a computation.



(a) MUL (b) ADD

Fig. 4. Histograms of MUL and ADD operands from neural network convolutional layers.

tend to cluster around some specific values. For example in MUL, about 15% of the operands appear near -1 and 6% - 7% operands appear near 0; for ADD, more than 30% of the operands fall into the interval around 0, which indicates there might be a considerable amount of trivial computations in CNNs.

To further explore the bypassing opportunities, we use two datasets: the hand-written *digits* MNIST [17] and the hand-written *letters* EMNIST [4]. Table II displays a percentage of trivial computations with relative to all computations in a CNN inference, where several observations can be made. First, for both datasets, the percentage of trivial computations exceed 50% of overall computations required. This suggests a good opportunity to perform computation bypassing in CNNs. Second, case (a) only accounts for approximately one-fourth of the overall trivial computations. While this number can vary depending upon different datasets and CNNs, the inclusion of other cases significantly increases the bypassing opportunity. Third, MUL operations present a higher percentage than ADD operations, which can be beneficial because MUL operations are more energy intensive. Additionally, we observe that case (e) presents a percentage less than 0.01%, which is significantly rare compared to the rest cases. Thus, we may consider removing hardware support for this case if its benefits cannot justify the extra overhead for detecting this case.

C. Trivial Computations Bypass

After exploration of trivial computations at the algorithm level, we design bypass circuits to detect and bypass selected cases of trivial computations. The bypass circuits should be able to detect all listed trivial computations in Table I. Thus, we detect the bypass circuits with the following logic:

MUL

TABLE II
PERCENTAGE OF TRIVIAL COMPUTATIONS IN NEURAL NETWORKS BY CASE.

	(a)	(b)	(d)	(e)
MNIST	15.35%	24.17%	21.02%	<0.01%
EMNIST	15.33%	20.76%	19.33%	<0.01%

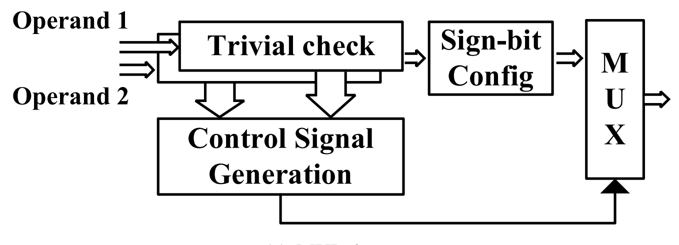
- Check if any of the two operands equals to 0 (case (a)). If so, the circuit will output bypassing result of 0 directly.
- If neither operand equals to 0, check if any of the two operands equals to ± 1 (case (b)). If so, the circuit will output its counterpart in this computation. Under this case, following the arithmetic rule, the circuit will also determine the sign-bit of the MUL computation bypassing result by operating XOR with the sign-bit from both operands.
- If this MUL computation is not trivial (i.e., case (c)), the bypass circuit will not output any result or clock-gate the FPU. The multiplexer will select the FPU channel and output the FPU result.

ADD

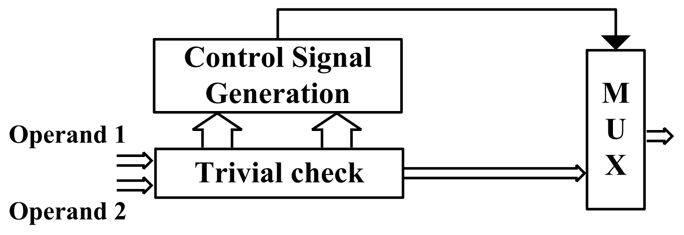
- Check if any of the two operands equals to 0 (case (d)). If so, the circuit will output its counterpart in this computation as bypassing result.
- If not, check if the two operands are inverse numbers of each other (i.e., case(e)). The bypass circuit will output 0 if so.
- If this ADD computation is not trivial (case (f)), the bypass circuit will not output any result or clock-gate the FPU. The multiplexer will select the FPU channel and output the FPU result.

The logic diagrams of the bypass circuits are illustrated in Fig. 5, where two main tasks are accomplished: detection of the trivial computations and generation of the corresponding control signal for the multiplexer, i.e., bypassing the corresponding trivial computations.

Accordingly, the gate-level hardware implementations of both MUL and ADD bypass circuits are shown in Fig. 6. To check if the computation is trivial, the circuit will first check if the bit representation of it aligns with that of trivial computation operands. According to the IEEE-754 standard for floating point number [1], one single precision (32-bit) floating point number consists of three fields: the 1-bit sign,

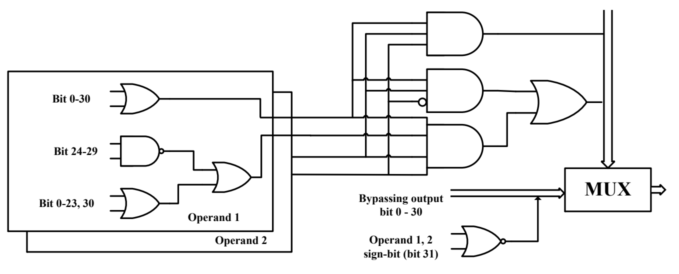


(a) MUL_bypass

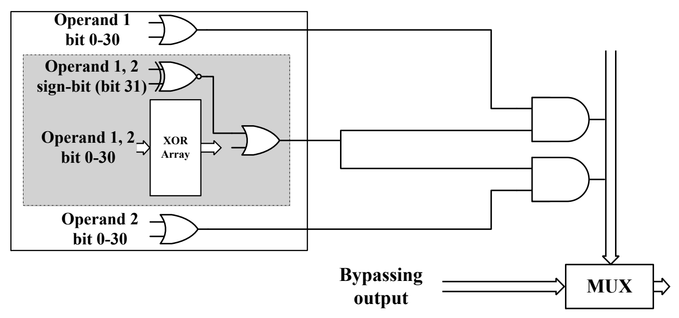


(b) ADD_bypass

Fig. 5. Diagram of MUL and ADD bypass circuits.



(a) MUL_bypass



(b) ADD_bypass

Fig. 6. Hardware implementation of MUL and ADD bypass circuits.

one 8-bit exponent field, and one 23-bit significand (mantissa) field. Therefore, the bit representation of operands from trivial computations could be described by Table III.

TABLE III
BIT REPRESENTATION OF OPERANDS FROM TRIVIAL COMPUTATIONS.

value		bit representation
0		0 00000000 00000000000000000000000
1		0 01111111 00000000000000000000000
-1		1 01111111 00000000000000000000000
$(a + b = 0)$	a	0(1) ......
	b	1(0) ......

all bits of a and b should be identical except for the sign bit.

To identify a trivial computation, however, does not require an intensive check or read on all 32 bits in an operand iteratively. For example, to identify ±1, three groups of gates are required as illustrated in Fig. 6(a): NAND gates with inputs from bit 24 - 29, OR gates with inputs from the rest bits and OR gates with the outputs of its preceding gates as inputs. Only if the input operand is ±1 will the gate output 0 for following circuits to generate control signals, indicating a successful identification of trivial computations. Similarly, both MUL and ADD bypass circuit will generate a 2-bit control signal for the multiplexer as Table IV presents.

TABLE IV
CONTROL SIGNALS WITH THEIR CORRESPONDING CASES.

control signal		paraphrase	
bit 1	bit 0	MUL	ADD
0	0	case (a)	case (e)
0	1	case (b), $b = \pm1$	case (d), $b = 0$
1	0	case (b), $a = \pm1$	case (d), $a = 0$
1	1	case (c)	case (f)

To output the result with correct sign-bit under case (b) in Table I, the MUL bypass circuit requires additional sign-bit configuration as well. For instance, the result of the trivial computation -1 $\times$ 0.75 (binary representation: 10111111100000000000000000000000 $\times$ 00111111010000000000000000000000) should be -0.75. The circuit will identify and bypass this trivial computation by outputting 0.75 then operating XOR with the sign-bits from the operands to subsequently modify it to the expected result -0.75.

However, as per our experimental results from Table II, case (e) of trivial bypassing is rather rare (less than 0.01%). Therefore, the step of checking case (e) can be skipped, the corresponding logic (the shading part in Fig. 6(b)) will also be omitted to save energy/area overhead of bypass circuits.

IV. EXPERIMENTAL RESULTS

A. Experimental Setup

We use tiny-dnn [22] to profile the input operands and computations in LeNet-5 like CNN architecture. Tiny-dnn is a portable (header only), dependency-free deep learning framework written in C++, of which low libraries can be readily modified to implement our methodology. We use MNIST [17] and EMNIST [4] as our datasets. The MNIST is a handwritten *digits* dataset with a training set of 60,000 and a test set of 10,000 examples. EMNIST is a hand-written *letter* dataset of 88,000 characters for training and 14,800 for testing from 37 classes [4].

We write the register-transfer level (RTL) descriptions of bypass circuits in Verilog. The Verilog codes are then synthesized using Synopsys Design Compiler and place-and-route using Synopsys IC Compiler. Synopsys PrimeTime is used to evaluate and measure the power and area of the circuits under 1.0V. The energy consumption of floating point adder and multiplier are adopted from [7], [25] under 1.0V.

B. Energy Savings

Eq. 1 describes how the total energy saving ε is calculated, where p_m and p_a refer to the percentage of MUL and ADD trivial computations respectively from Table II; E_{fm} and E_{fa} refer to energy consumption of each MUL and ADD float computation respectively by FPU from Table V; E_{bm} and E_{ba} refer to energy consumption of bypassing each trivial computation respectively from Table V.

$$\varepsilon = \frac{p_m(E_{fm} - E_{bm}) + p_a(E_{fa} - E_{ba})}{E_{fm} + E_{fa}} \tag{1}$$

For each case among the trivial computations, the energy saving ε_i could be acquired from Eq. 2, where p_i refers to the percentage of the trivial computation of this specific case; E_{fi} and E_{bi} refer to the energy consumption of float computation of this case by FPU and bypassing, respectively.

$$\varepsilon_i = \frac{p_i(E_{fi} - E_{bi})}{E_{fm} + E_{fa}} \tag{2}$$

According to our observation, the bypass circuit for case (e) (the shading part from Fig. 6(b)) can be omitted due to its extremely low percentage. Thus the energy consumption of bypassing each ADD trivial computation can be further lowered to 12.2 FJ from 23.7fJ, as shown in Table V.

TABLE V
ENERGY CONSUMPTION OF DIFFERENT COMPUTATIONS.

computation		energy (FJ)
FPU	MUL	9891
	ADD	4742
Bypass Circuits	MUL	12.5
	ADD	23.7
	ADD w/o shading part	12.2

Fig. 7 illustrates the energy-saving of the three feasible cases obtained from Eq. 1 and Eq. 2, where several observation can be made. First, the proposed method can lead to 30.66% and 33.52% energy savings over baseline architecture on EMNIST and MNIST without any accuracy loss, respectively. Second, energy saving from MUL computations dominates that from ADD, and contributes most to the total energy-saving we have obtained. This is because of two reasons: MUL naturally consumes more energy than ADD as shown in Table V; the percentage of trivial computations related to MUL is significantly higher than that of ADD. Third, the energy savings induced by case (a), i.e., ineffectual multiplications that only consider zero-aware multiplications [2], [15], [14], are only 10.78% and 10.37% for EMNIST and MNIST. While our proposed approach could take advantage of more trivial computations, i.e., case (b) and case (d), and thus could extend the energy reduction 3X more to 30.66% and 33.52%. Actually, case (b) (i.e., multplication with operand +1/-1) contributes most of the energy benefits. This suggests the

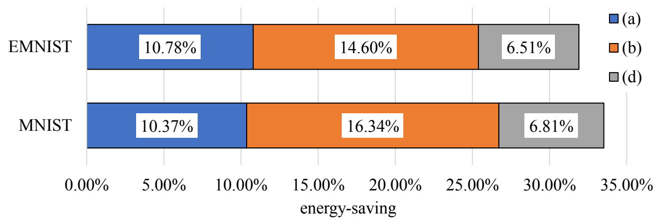


Fig. 7. Energy-saving contribution breakdown by case.

importance of including these trivial multiplications. To our best knowledge, we are the first one to consider and utilize such computation property. We also measure the area cost induced by the bypass circuits with respect to a FPU: 2.7% for adder and 3.1% for multiplier.

C. Discussion

Potential Usage In addition to energy-saving, bypassing trivial computations can also accelerate neural network inferences because the bypass circuits have lower latency than the floating point units. Since the exact acceleration effects depend upon concrete architecture and how the parallelism are implemented, this paper does not evaluate such effects because this paper focuses on measuring the architecture-independent energy benefits of bypass circuits. Actually, the bypass circuits can be applied to any hardware architecture such as GPU, ASIC, or FPGA with floating point units.

Potential Improvement The percentage of trivial computations can be further improved by using *approximate operands matching*. For example, we can treat an operand that is close enough to 0 or 1 as 0 or 1. This is motivated by Fig. 4 where the operands tend to cluster around some specific values indicating trivial computations. Potential approximate matching can be completed by truncating the bit length required for operand matching or using Hamming Distance-based binary vector matching. However, this method can lead to potential accuracy loss so we leave it for future works.

V. Related Work

Many kinds of specialized accelerators have been proposed for accelerating neural network algorithms such as CNN [9], [10], [19], [8], [21], [2], [15], [14], [13].

Early success of hardware accelerators leverages pruning-based techniques to shrink the original dense network size by removing the neuron connections with weights below a certain threshold, which leads to a sparse network structure [9], [10]. Another set of works use quantization that converts the floating point operations into fixed point operations to reduce the computation workload of CNNs [19], [8]. While effective, these methods typically require *retraining* or fine-tuning to regain the accuracy, which lacks flexibility and can be time-consuming. Even using retraining, these methods can incur accuracy loss.

A promising approach is to exploit the *ineffectual multiplications* in CNNs [2], [15], [14]. Ineffectual multiplications are defined as multiplication where one of the its input operands is zero (zero-aware multiplications). These multiplications can be skipped because if any of the two operands is zero, the final product can be automatically determined as zero without actual operations. Therefore, by simply skipping these ineffectual multiplications, it is possible to reduce the computation workload without any accuracy loss. Various works propose to skip these ineffectual multiplications to reduce computation workload [15], [14]. Furthermore, the ineffectual multiplications can be combined with network pruning, where the near-zero neurons are also set to zero when their magnitudes are below a certain threshold [2]. This leads to a more aggressive computation skipping, where the performance is improved at the cost of accuracy loss. However, in this paper, we have shown that the portion of ineffectual multiplications only accounts for approximately one-fourth of the overall trivial computations.

Our approach is different from these aforementioned works in several aspects: 1) Our approach is retraining-free and can still preserve the original network accuracy. 2) Our work utilizes the data locality of multiplications as well as additions. This extra coverage leads to enhanced bypass opportunities. 3) Our approach is totally complementary to existing CNN acceleration techniques such as pruning and compression so they can be combined to provide extra benefits.

VI. Conclusion

The high computational complexity of neural networks remains a compelling challenge for their deployment in hardware. In this paper we present an approach to reducing the computation workload of convolutional neural networks through detecting and bypassing trivial computations of which results can be determined without actual floating point operations. We classify trivial computations into various cases based on their input operands. We then design bypass circuits to detect and bypass trivial computations. Without any accuracy loss, the proposed approach can reduce energy consumption by 30.66-33.52% on MNIST and EMNIST. Our future work will focus on exploring the accuracy-energy trade-offs under different approximate operand matching schemes, using techniques such as truncated bit-length matching.

References

[1] Ieee standard for floating-point arithmetic. *IEEE Std 754-2008*, pages 1–70, Aug 2008.

[2] Jorge Albericio, Patrick Judd, Tayler Hetherington, Tor Aamodt, Natalie Enright Jerger, and Andreas Moshovos. Cnvlutin: Ineffectual-neuron-free deep neural network computing. In *ACM SIGARCH Computer Architecture News*, volume 44, pages 1–13. IEEE Press, 2016.

[3] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*, pages 367–379. IEEE, 2016.

[4] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and André van Schaik. EMNIST: an extension of MNIST to handwritten letters. *CoRR*, abs/1702.05373, 2017.

[5] Zidong Du, Avinash Lingamneni, Yunji Chen, Krishna V Palem, Olivier Temam, and Chengyong Wu. Leveraging the error resilience of neural networks for designing highly energy efficient accelerators. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(8):1223–1235, 2015.

[6] Andre Esteva, Brett Kuprel, Roberto A Novoa, Justin Ko, Susan M Swetter, Helen M Blau, and Sebastian Thrun. Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542(7639):115, 2017.

[7] Amirali Ghofrani, Abbas Rahimi, Miguel A Lastras-Montaño, Luca Benini, Rajesh K Gupta, and Kwang-Ting Cheng. Associative memristive memory for approximate computing in gpus. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 6(2):222–234, 2016.

[8] Philipp Gysel. Ristretto: Hardware-oriented approximation of convolutional neural networks. *arXiv preprint arXiv:1605.06402*, 2016.

[9] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. Eie: efficient inference engine on compressed deep neural network. In *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*, pages 243–254. IEEE, 2016.

[10] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[12] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[13] Xun Jiao, Vahideh Akhlaghi, Yu Jiang, and Rajesh K Gupta. Energy-efficient neural networks using approximate computation reuse. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1223–1228. IEEE, 2018.

[14] Dongyoung Kim, Junwhan Ahn, and Sungjoo Yoo. A novel zero weight/activation-aware hardware architecture of convolutional neural network. In *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1462–1467. IEEE, 2017.

[15] Dongyoung Kim, Junwhan Ahn, and Sungjoo Yoo. Zena: Zero-aware neural network accelerator. *IEEE Design & Test*, 35(1):39–46, 2018.

[16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

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

[18] Yann LeCun et al. Lenet-5, convolutional neural networks. *URL: http://yann. lecun. com/exdb/lenet*, page 20, 2015.

[19] Darryl Lin, Sachin Talathi, and Sreekanth Annapureddy. Fixed point quantization of deep convolutional networks. In *International Conference on Machine Learning*, pages 2849–2858, 2016.

[20] Vojtech Mrazek, Syed Shakib Sarwar, Lukas Sekanina, Zdenek Vasicek, and Kaushik Roy. Design of power-efficient approximate multipliers for approximate artificial neural networks. In *2016 International Conference On Computer Aided Design (ICCAD)(prijato)*, page 7, 2016.

[21] Hardik Sharma, Jongse Park, Naveen Suda, Liangzhen Lai, Benson Chau, Vikas Chandra, and Hadi Esmaeilzadeh. Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 764–775. IEEE, 2018.

[22] tiny dnn. https://github.com/tiny-dnn/tiny-dnn.

[23] Swagath Venkataramani, Ashish Ranjan, Kaushik Roy, and Anand 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.

[24] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1492–1500, 2017.

[25] Xunzhao Yin, X Sharon Hu, et al. Design and benchmarking of ferroelectric fet based tcam. In *Proceedings of the Conference on Design, Automation & Test in Europe*, pages 1448–1453. European Design and Automation Association, 2017.