Reducing MAC operation in convolutional neural network with sign prediction

Jiho Chang
Intelligent Robot System Research Group
Electronics and Telecommunications Research Institute
Daejeon, Korea
changjh@etri.re.kr

Yoonsung Choi, Taegyoung Lee, Junhee Cho
School of Computing
Korea Advanced Institute of Science and Technology
Daejeon, Korea
{giantsol2, taegyoung, junheecho}@kaist.ac.kr

***Abstract*—Due to recent researches on artificial neural network algorithms and machine learning, the accuracy of image recognition and natural language processing has increased to the level of human beings in specific fields. Especially, researches to improve the accuracy of algorithms are being actively conducted, and researches on hardware accelerators that implement such algorithms quickly and efficiently are actively under way. In order to utilize artificial intelligence reasoning ability as well as computing speed in mobile or embedded environment, it is necessary to reduce the power consumption and memory usage of artificial intelligence hardware. In this paper, we propose a algorithm to reduce the computational complexity in designing the CNN accelerator. We tried to reduce the MAC computation by encoding the inputs and predicting the sign of the MAC operation. We confirmed the performance improvement by evaluating the sign predictor through the simulation results.**

***Index Terms*—CNN, Hardware accelerator, Sign prediction, activation function**

I. Introduction

Recently, deep neural networks (DNN) have been widely adopted in many computer vision areas as image processing, medical imaging, activity recognition, and robotics. [7] This is because performance of parallel process calculators has been improved to handle a large number of computations and weights for DNN. [4], [8] General-purpose graphics processing unit (GPGPU) has been prevalently used to accelerate neural network computations in both academia and industry. Because GPGPU has been introduced for intensive parallel computation in computer graphics, it is inherently apt for neural network computations, which are easily parallelized. In general, GPGPU is good for parallel computations; however, it is not tailored for neural network computations. The large energy consumption and the large size of GPGPU are pivotal issues. Especially, in mobile platforms such as drone, robot, and mobile phone, it is hard to embed GPGPUs. Hence, many research results searched for an accelerator which outperforms GPGPU in neural network computations in terms of performance per energy consumption. There are already several research results to deal with these issues by different approaches: reducing the size of data storage; the bus usage to transmit in and out of processing units; the network connections by pruning; and the size of data i.e. weights to make the size of processing unit small.

This work was supported by the ICT R&D program of MSIP/IITP. [2017-0-00306, Development of Multimodal Sensor-based Intelligent Systems for Outdoor Surveillance Robots]

Hardware-based accelerations of DNNs usually take two approaches. The first approach is to reduce the number of multiply-accumulate (MAC) operations because accelerators are designed to be small and thus the number of processing elements (PE) is small unlike GPGPU with thousands of cores. The second approach is to make memory access more efficient. Caching is difficult because a cache entry is not repeatedly used as much as in general applications. Meanwhile, PE scheduling and careful configurations of board aims to reduce the workload on bottlenecks in the memory bus.

We focus on reducing the number of MAC operations in convolution layer (CONV), which is involved in convolutional neural networks (CNN), generative adversarial network (GAN), and autoencoder. *CNN* is a feed-forward DNN which has long been used in image processing, video analysis, natural language processing, and even in Go. *GAN* is a system of two neural networks competing each other in a zero-sum game framework. It has been used in image recognition, pattern recognition from a video, reproducing photorealistic image and video. *Autoencoder* is a neural network to learn generative models of data and an encoding for a set of data. The goal of training of autoencoder is to find an encoding where data is compressed and then uncompressed to another data which closely matches to the original data. Thus, it has been widely used in dimensionality reduction.

In this background, we will survey the state-of-art DNN accelerators and try improving them by reducing the number of MAC operations as described in Section III.

A. Trends in CNN

Convolutional Neural Network (CNN) is one of widely used neural network and cannot be neglected. It mainly consists of convolution layer (CONV) and fully-connected layer (FC). They contrast how they compute and how they access memory. The number of parameters and computations are shown in Table I. Fully-connected layer holds most of parameters, i.e., weights. Although CNN usually consists of a few fully-connected layers and they hold a few of computations, they

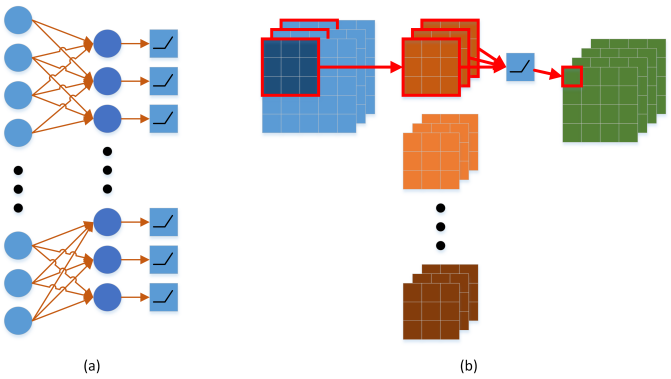

Fig. 1. Concept diagram of fully connected layer and convolution layer

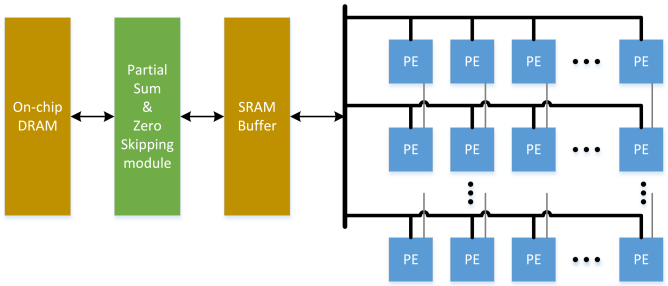

Fig. 2. Achitecture of EYERISS

are the bottleneck to degrade performance because they access memory intensively. On the other hand, convolution layers which hold most of computations tend to have increasing number of parameters.

Figure 1 conceptually illustrates the computation of a fully connected network and a convolution network. Since FC process the calculation for all connections to the input, the weight and the number of operations increase in proportion to the input. In case of CNN, the input of convolution layer are 2D arrays and the output are again 2D arrays. The computations are 2D convolutions sliding over the input arrays. *Kernel* or *filter* is a 2D array of weights. It is repeatedly used as it convolve over the input. Therefore, CNN has relatively less weights than FCN, but it requires a lot of iterative operations. Neural network accelerators reduce the number of computations or the number of memory accesses for that reason. For instance, accelerators targeting GooLeNet and Resnet-152, which are increasingly used, would reduce the number of computations in convolution layers. [9] [6]

II. Related Works

In this section, we survey recent research results on energy efficient accelerators for neural networks.

a) EYERISS: EYERISS proposed by Stanford Univ. [2] is a neural network accelerator consists of a 2D array of PEs with a local register file, and a global SRAM buffer shared by all PEs. Figure 2 shows the top-level architecture and memory hierarchy of the EYERISS. EYERISS is designed to enable the scheduler to select the data to be executed at each PE and to pass the data through the connected bus between the PEs in order to use the accelerator efficiently. For such computation, each PE can either access local memory or communicate with neighbor PEs or the SRAM buffer. EYERISS has two approaches to improve the energy efficiency: reducing data transmission and exploiting data statistics. So, it has run-length encoding codec, and zero skipping module with ReLU.

b) Tetris: Tetris is based on accelerators with the same structure as EYERISS to reduce energy consumption by making scheduling more efficient. Scheduling is particularly important for using small buffers which could access to slow DRAM. Tetris [3] schedules NN layer of arbitrary sized onto the given size physical PE array to maximize data reuse. Furthermore, it supports a tiled architecture with multiple nodes that can partition and process computations in parallel. Each node is an Eyeriss-style engine. While it is used by users, user can specify NN batch size and word size, PE array dimensions, number of tile nodes, register file and global buffer capacity, and the energy cost of all components. Note that the energy cost of array bus should be the average energy of transferring the data from the buffer to one PE, not local neighbor transfer.

A. Reducing the number of operations in convolution layers

In convolutional layers in most of frequently used CNNs, each convolution operation is commonly followed by an activation function called a Rectifying Linear Unit (ReLU). ReLU function returns zero for negative inputs and returns the input value back for the positive ones. Vahideh et al. observe that a large amount of ReLU outputs are zero, due to a large number of negative convolution outputs. [1] Also, they show this trend among several popular CNNs where ReLU sets 42%-68% of outputs to zero. Therefore, by utilizing the characteristics of the ReLU, it is possible to spare a large number of MAC operations, to reduce memory access, and to keep small space of the intermediate result storage.

In the EYERISS NN accelerator, they use the output activations of the feature maps are sparse after the ReLU. The sparsity can be optimized for energy and area savings using data compression, particularly for slow DRAM access. In addition to compression, the accelerator skips reading the weights and the inputs to perform the MAC for zero-valued activations to reduce energy cost by 45%. However, EYERISS NN accelerator reduces the amount of the following convolution calculations, but must calculate them at the beginning of feature input and weight input.

On the other hand, the SnaPEA proposed by Vahideh et al. rearranges the ordering of inputs and weights according to the sign of the weight, and predicts the sign of output value as the accumulation is processing. To reduce the computations

TABLE I
COMPARISON OF CONVOLUTION LAYER (CONV) AND FULLY-CONNECTED LAYER (FC) IN TERMS OF COMPUTATION

Model	Parameters (million)	CONV (%)	FC (%)	Operations (million)	CONV (%)	FC (%)
AlexNet	61.0	3.8	96.2	725	91.9	8.1
VGG-16	138.0	10.6	89.4	15,484	99.2	0.8
GoogLeNet	6.9	85.1	14.9	1,566	99.9	0.1
ResNet-50	25.6	-	-	3,900	-	-

further, SnaPEA speculates on the sign of the outputs before starting with the negative weights. The proposed method shows that if the partial output of MAC operations is less than a threshold, we predict the final convolution output will be negative. Vahideh et al. insist their accelerator has 28% speedup and 16% energy reduction in various modern CNNs than a EYERISS CNN accelerator without affecting their classification accuracy.

B. Network quantization and weight sharing

Recent research results compress neural networks by network quantization and weight sharing. They reduce the number of effective weights by sharing one of similar weights with the others. In linear approach, floating point numbers are transformed into fixed point numbers. This approach is employed in commercial accelerators such as tensor processing unit (TPU) introduced by Google and NVIDIA PASCAL GPU. However, this approach results a degraded quality as the bit size is reduced because of wide range of weights. [5] In non-linear approach, which consider non-uniform distribution of weights, non-linear quantization results less loss of quality. However, a hardware implementation of non-linear quantization is difficult.

III. PROPOSED ALGORITHM

Between two approaches introduces in Section II, our approach is to reduce the number of operations, MAC operations. We encode the weights and the values from the input layer in fixed or floating point number in reduced size. Then, we evaluate the MAC operation with the numbers in reduced size and predict the sign of the MAC operation with the original numbers. The MAC operations with numbers in reduced size runs faster than the MAC operations with the original numbers. If the prediction is correct, we safely skip the MAC operations resulting the output 0 without loss of quality. Figure **??** and 3 illustrates the overview of our approach and the sign prediction unit signals a termination control signal to terminate the MAC operation with the original numbers when a negative number is predicted.

In Subsection III-A, we introduce the encoding and prove the correct of prediction. In Subsection III-B, we introduce an encoding more suitable to implement in hardware.

A. Encoding weight and input

In this section, we introduce a hardware-friendly encoding which does not produce a loss.

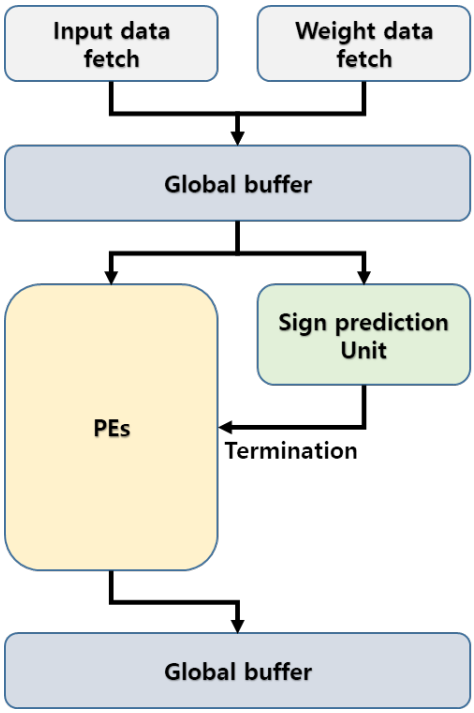

Fig. 3. Flow chart of termination

Let $\vec{p} = (p_1, p_2, \cdots, p_n)$ and $\vec{q} = (q_1, q_2, \cdots, q_n)$. Then, our goal is to predict whether $\vec{p} \cdot \vec{q} = \sum_i^n p_i q_i$ is positive or negative. We round all components p_is and q_is from the fifth digit from the first digit with 1. Let $\vec{r} = (r_1, r_2, \cdots, r_n)$ and $\vec{s} = (s_1, s_2, \cdots, s_n)$ be the rounded numbers. For example, if $\vec{p} = 0.001101001$ and $\vec{q} = 0.0100110$, $\vec{r} = 0.001101$ and $\vec{s} = 0.01010$. Then, for each i, $|p_i - r_i| \leq 2^{-x_i}$ and $|q_i - s_i| \leq 2^{-y_i}$ for some x_i and y_i.

Lemma III.1. $|p_i q_i - r_i s_i| \leq 2^{-x_i}|s_i| + 2^{-y_i}|r_i| + 2^{-x_i - y_i}$ *for all* i.

Proof. By the conditions,

$$r_i - 2^{-x_i} \leq p_i \leq r_i + 2^{-x_i} \quad (1)$$

$$s_i - 2^{-y_i} \leq q_i \leq s_i + 2^{-y_i} \quad (2)$$

If p_i and q_i are positive,

$$(r_i - 2^{-x_i})(s_i - 2^{-y_i}) \leq p_i q_i \leq (r_i + 2^{-x_i})(s_i + 2^{-y_i}) \quad (3)$$

If p_i and q_i are negative,

$$(r_i - 2^{-x_i})(s_i - 2^{-y_i}) \geq p_i q_i \geq (r_i + 2^{-x_i})(s_i + 2^{-y_i}) \quad (4)$$

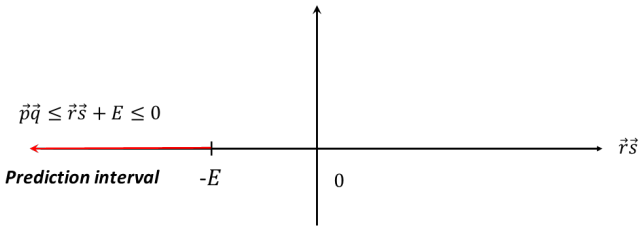

Fig. 4. Prediction interval

Now, the remaining case is when one of p_i and q_i is positive and the other is negative. Let p_i is positive and q_i is negative. Then,

$$(r_i + 2^{-x_i})(s_i - 2^{-y_i}) \le p_i q_i \le (r_i - 2^{-x_i})(s_i + 2^{-y_i}) \quad (5)$$

On the right-hand side inequality,

$$\begin{aligned} p_i q_i &\le r_i s_i - 2^{-x_i} s_i + 2^{-y_i} r_i - 2^{-x_i - y_i} && (6) \\ &\le r_i s_i + 2^{-x_i} s_i + 2^{-y_i} r_i + 2^{-x_i - y_i} && (7) \end{aligned}$$

Similarly, on the left-hand side inequality,

$$\begin{aligned} p_i q_i &\ge r_i s_i - 2^{-x_i} s_i + 2^{-y_i} r_i - 2^{-x_i - y_i} && (8) \\ &\ge r_i s_i - 2^{-x_i} s_i - 2^{-y_i} r_i - 2^{-x_i - y_i} && (9) \end{aligned}$$

Therefore, it holds in all cases. □

With Lemma (III.1),

$$|\vec{p} \cdot \vec{q} - \vec{r} \cdot \vec{s}| \le \sum 2^{-x_i} |s_i| + \sum 2^{-y_i} |r_i| + \sum 2^{-x_i - y_i} \quad (10)$$

In other words,

$$\vec{p} \cdot \vec{q} \le \vec{r} \cdot \vec{s} + \Big(\sum 2^{-x_i} |s_i| + \sum 2^{-y_i} |r_i| + \sum 2^{-x_i - y_i} \Big) \quad (11)$$

Therefore, $\vec{p} \cdot \vec{q} \le 0$ when the following holds

$$\vec{r} \cdot \vec{s} \le -\Big(\sum 2^{-x_i} |s_i| + \sum 2^{-y_i} |r_i| + \sum 2^{-x_i - y_i} \Big) \quad (12)$$

We compute the both sides of Equation (12) and compare the two values. If Equation (12) holds, we conclude the result of the MAC operation $\vec{p} \cdot \vec{q}$ is negative and skip the MAC operation.

This prediction is correct: no false-positive. If the original weighted sum is positive, the weighted sum in our encoding is positive. If the original weighted sum is negative, the weighted sum in our encoding might be positive or negative. Therefore, our predictor reports negative only when the original weighted sum is negative.

More precisely, our predictor reports negative when the original weighted sum is less than the right-hand side of Equation (12). Let $-E$ be the right-hand side of Equation (12). The prediction interval is illustrated in Figure 4. When we use less bits in the encoding, the running time of prediction would decrease, but E becomes larger. When we use more bits in the encoding, we can make E smaller, but the running time of prediction would increase.

B. Hardware-oriented encoding

The prediction introduced in Subsection III-A involves finding the position of first bit with value 1. It is occasionally implemented by series of shift operations, which requires multiple clock cycles. Because its output timing depends on the value of input, a complicated logic should be implemented on hardware. When we use fixed point encoding instead of floating point encoding, the output timing of prediction is fixed; thus, it is easy to implement on hardware. For example, if $\vec{p} = 0.001101001$ and $\vec{q} = 0.0100110$, we use $\vec{r} = 0.0011$ and $\vec{s} = 0.0101$ instead of $\vec{r} = 0.001101$ and $\vec{s} = 0.01010$. Figure 5 is a block diagram for our hardware oriented archtecture.

IV. Planned methodology

A. Evaluation metric

The MAC operation computes the product of two numbers and adds that product to an accumulator. Generally, CNN processing consists of 3 layers that are convolution layer, pooling layer and fully connected layer. Convolution layer which consume the most time of execution time is made up basic MAC operation. Therefore, high speed accelerators choose specific structure that accelerate MAC operation in parallel using a large amount of MAC operation. In order to do MAC operation with high speed, input feature map information and filter weight information of current layer are read on each memory. And gathering information are processed by PE(Processing element) based on MAC operation and then output feature map information for next layer are restored on memory. Therefore, we measure the number of MAC operations performed by one method for evaluation between algorithms. Since the scheduling tools count the number of operations for each layer, various workloads and algorithms can be evaluated. Energy efficiency is measured by the throughput per watt and it is determined by energy consumption. We are expecting less energy consumption than others.

B. Workloads

We use several famous NN workloads to evaluate our algorithm. AlexNet, invented by Alex Krizhevsky, is famous neural network won the 2012 ImageNet LSVRC-2012 competition by a large margin. alexnet is designed to fit image classification, but it is known to express feature information about images well, and it is also used nowadays because it is not heavy network compared to modern NN.

V. Simulation and results

In this section, we will perform simulation by inputting randomly generated values and real images to alexnet for evaluating negative prediction accuracy when using the proposed algorithm.

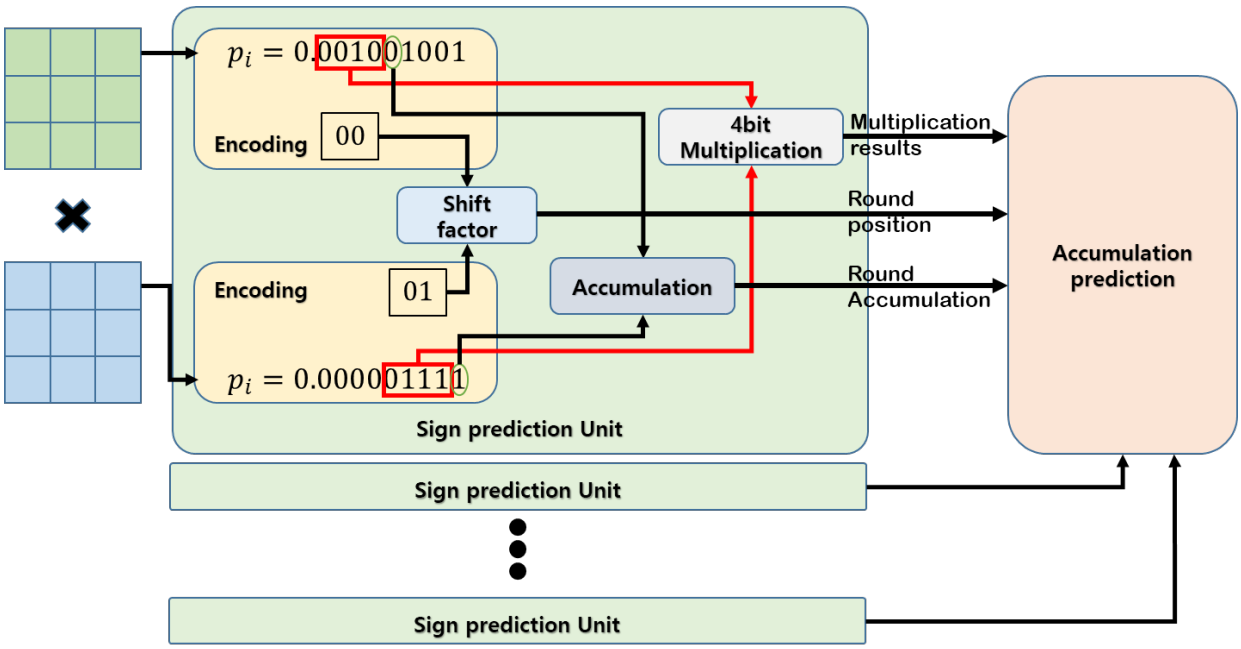

Fig. 5. Overview of our hardware oriented architecture

A. Random generation results

In order to verify the effectiveness of various networks and fully connected layers other than specific CNNs, weight and input are generated as random variables with specific distribution and the results of the sign prediction of the MAC operation are confirmed. Weight is generated to follow the gaussian distribution when most CNNs are well trained, and input is generated to follow the uniform distribution. Both data assume a 16-bit fixed point, which is the size of data that conventional CNN accelerators typically used. Assuming 1000 sums of vector products multiplied by size of 300 and 300, we performed 10 runs for each encoded bit. Figure 6 shows the average of the predicted ratio for the negative result of the original multiplication operation depending on size of encoding bit. In the case of 12bit, 100% of the negative number was predicted. As the bit decreased, the prediction rate decreased, and the accuracy of prediction was about 80% (95% or more in case of HW) in 4bit.

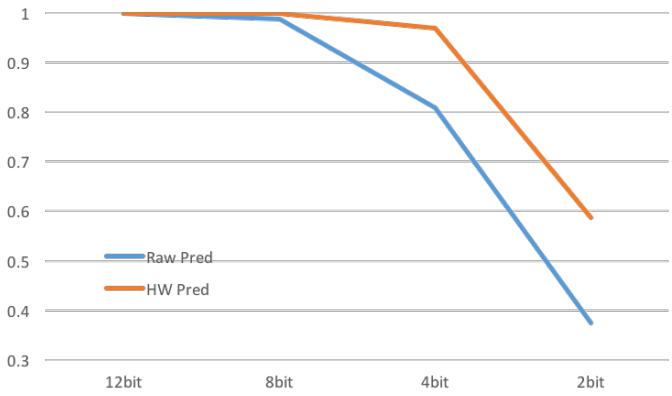

Fig. 6. Prediction results for random values

B. AlexNet results

In the case of real networks, since the weight value is learned, it has a biased shape instead of a perfect gaussian distribution. Therefore, we conducted experiments using AlexNet learned for image classification to determine how negative prediction actually occurs in CNN. For the two image inputs, 4-bit encoding of the HW oriented method is used, and the following Table II shows the results in the 1st layer. Among the results of the total convolution, 151267 (laska) and 152960 (poodle) negative numbers occur, and it can be confirmed that most of the negative numbers can be predicted as 88% (laska) and 77% (poodle). It accounts for about 45% of the total MAC operation in the convolution layer, and it can be seen that it can reduce a considerable amount of computation.

VI. Conclusion and Future Work

In this paper, we propose a method to improve performance by interrupting through the sign prediction of MAC operation. For this algorithm, we showed mathematically satisfactory results for a new encoding method with a low bit size, and proposed a method suitable for hardware implementation. The original plan was to simulate the MAC operation in the Tetris simulator to measure the energy and time that would actually be reduced, but the simulator had difficulty changing because it is an implementation of a static analysis calculating from the network structure without actually running the neural network

TABLE II
PREDICTION RESULTS FOR ALEXNET

Input	Real ≤ 0	Pred ≤ 0	Pred > 0
Laska	151,267	133,209 (88.06%)	18,058 (11.94%)
Poodle	152,960	118,899 (77.73%)	34,061 (22.27%)

computations rather than a dynamic analysis which runs time-based operations. In the future, we will evaluate the overall energy consumption and the degree of saving of the MAC performance according to the timing of the termination signal result by completing the simulation including to the timing.

REFERENCES

[1] Vahideh Akhlaghi, Amir Yazdanbakhsh, Kambiz Samadi, Rajesh K. Gupta, and Hadi Esmaeilzadeh. Snapea: Predictive early activation for reducing computation in deep convolutional neural networks. *ISCA 2018*, Jun 2018.

[2] Y. H. Chen, T. Krishna, J. S. Emer, and V. Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits*, 52(1):127–138, Jan 2017.

[3] Mingyu Gao, Jing Pu, Xuan Yang, Mark Horowitz, and Christos Kozyrakis. Tetris: Scalable and efficient neural network acceleration with 3d memory. *SIGARCH Comput. Archit. News*, 45(1):751–764, April 2017.

[4] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *CoRR*, abs/1510.00149, 2015.

[5] Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems*, pages 1135–1143, Cambridge, MA, USA, 2015. MIT Press.

[6] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, June 2016.

[7] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521:436 EP –, 05 2015.

[8] H. Sharma, J. Park, D. Mahajan, E. Amaro, J. K. Kim, C. Shao, A. Mishra, and H. Esmaeilzadeh. From high-level deep neural models to fpgas. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1–12, Oct 2016.

[9] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, June 2015.