Efficient SIMD Implementation for Accelerating Convolutional Neural Network

Sung-Jin Lee
Dept. of Electronics Engineering
Hangdang-dong, Seongdong-gu
Seoul, Korea
82-2-2220-4701
leesky601@hanyang.ac.kr

Sang-Soo Park
Dept. of Electronics Engineering
Hangdang-dong, Seongdong-gu
Seoul, Korea
82-2-2220-4701
po092000@hanyang.ac.kr

Ki-Seok Chung
Dept. of Electronics Engineering
Hangdang-dong, Seongdong-gu
Seoul, Korea
82-2-2220-4701
kchung@hanyang.ac.kr

ABSTRACT

Convolutional Neural Network (CNN) has been used in a variety of fields such as computer vision, speech recognition, and natural language processing. Because the amount of computation has increased tremendously, CNN has lately been accelerated through accelerators such as Graphic Processing Unit (GPU). However, resource-constrained embedded platforms such as Internet of Things (IoT) devices cannot afford to have such accelerators. Therefore, it is important to accelerate CNN by only the CPU efficiently. In this paper, we propose a method to accelerate CNN by using the Single Instruction Multiple Data (SIMD) unit integrated in many CPUs. Modern CPU includes a SIMD unit which is commonly used for vector operations. The proposed method implemented on an ARM's NEON can maximize the utilization of vector registers in the SIMD unit. Our proposed implementation has achieved a speed-up of up to 2.66 in execution time and an energy reduction of up to 3.55 times than the conventional implementation.

CCS Concepts

•**Computer systems organization→Single instruction, multiple data;** Neural networks; Embedded systems.

Keywords

CNN; SIMD; NEON; LeNet-5; Parallel Processing; CPU Acceleration; OpenMP;

1. INTRODUCTION

Convolutional Neural Network (CNN), a type of Artificial Neural Network (ANN), is a neural network inspired by biological organization of the human optic nerve. CNN has been used in a variety of fields such as image, speech, and natural language processing through excellent recognition capability [1-3].

The deeper the depth of layers gets, the higher CNN's object-recognition accuracy gets. Therefore, there are various attempts to increase the number of layers. Correspondingly, the amount of computation increases massively in order to achieve better classification capability [4-5]. Therefore, CNN may have to require high-performance processing units such as Graphics Processing Units (GPU).

However, resource-constrained embedded platforms such as Internet of Things (IoT) devices cannot afford to have such accelerators because they operate with batteries with limited driving capability. Therefore, it is important to accelerate CNN by only the CPU efficiently.

Single Instruction Multiple Data (SIMD) units such as Intel AVX [6] and ARM NEON [7] are commonly integrated in modern Central Processing Units (CPUs), and they are utilized to accelerate media and data streaming [8]. The SIMD unit is capable of performing vector operations which carry out multiple operations of a single type in parallel. It also specializes in processing Multiply-Accumulate (MAC) operations because MAC operations are heavily executed in image processing and machine learning [9-10].

In this paper, we propose a technique to accelerate CNN with SIMD processing units. A CNN called LeNet-5, which is widely used in the Optical Character Recognition (OCR) field [11], is accelerated by utilizing SIMD instructions in NEON, the SIMD processing unit inside ARM CPUs. The NEON technology was first introduced in the ARMv7 architecture and it has been available with ARM Cortex-A class processors.

The SIMD lane is the space in which the data element is assigned to a vector register. One data element is assigned to one SIMD lane. When optimizing CNN with SIMD instructions in conventional methods, the SIMD lanes are not efficiently utilized because most convolution kernels have an odd size in both widths and heights while the number of SIMD lanes in a vector register is even. Therefore, we propose a method called Depth-Directional Method (DDM) to vectorize convolution kernels in the depth direction. In most CNN models, convolution kernels have an even depth. In general, each convolution layer has distinct characteristics. Therefore, in the proposed DDM, layer-specific SIMD optimization methods are applied. As shown in Figure 1, LeNet-5 consists of 3 convolution layers, 2 sub-sampling layers, and 2 fully-connected layers. First, in Convolution Layer 1 (C1), 2D convolution kernels are used while in Convolution Layer 2 and 3 (C2 and C3), 3D convolution kernels are used. We propose a method that minimizes the number of idle lanes to fully utilize the SIMD processing units. The proposed DDM in LeNet-5 achieves a speed-up of 3.45 in C1, 2.97 in C2, 3.32 in C3 by efficiently utilizing the SIMD resource.

ICCIP 2018, November 2–4, 2018, Qingdao, China
© 2018 Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-6534-5/18/11…$15.00
http://doi.org/10.1145/3290420.3290444

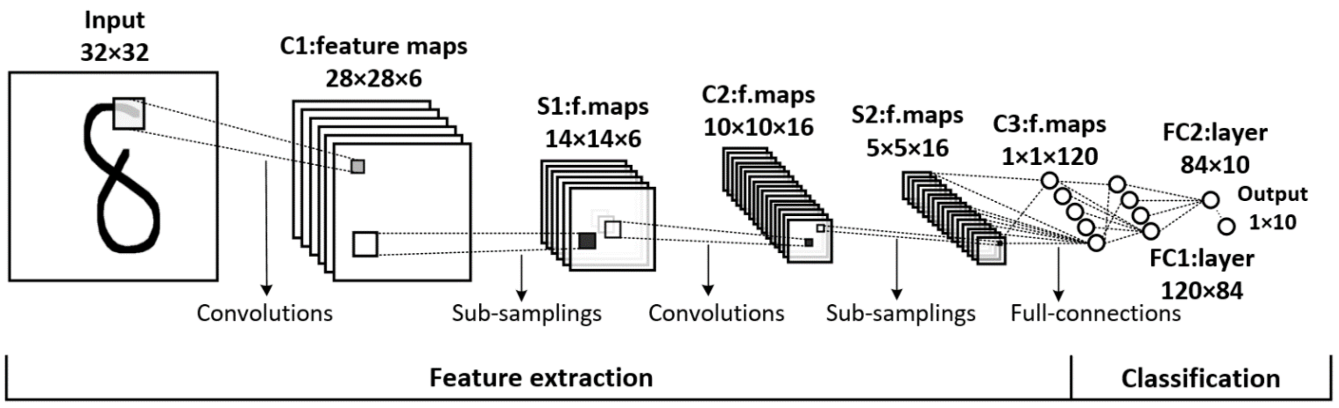

Figure 1. The architecture of LeNet-5

The rest of paper is organized as follows. Section 2 introduces CNN and our target CNN model called LeNet-5. Section 3 presents an introduction of SIMD in NEON and describes the proposed method. Section 4 shows our experiment results. Section 5 concludes this paper.

2. CONVOLUTIONAL NEURAL NETWORK

CNN, which is specialized in computer vision, is a model that mimics human vision processing. CNN is superior in image recognition among deep running techniques. CNN has attracted a lot of attention with overwhelming performance in image recognition [12]. CNN typically consists of two processes, feature extraction and classification. Feature extraction with convolutions and sub-samplings is a process to extract features of an object such as lines and edges. Classification with full-connections selects an object of the most probable category based on the extracted features. LeNet-5 consists of three convolution layers and two sub-sampling layers for the feature extraction process of an input image and two fully-connected layers for the classification process.

2.1 Convolution layer

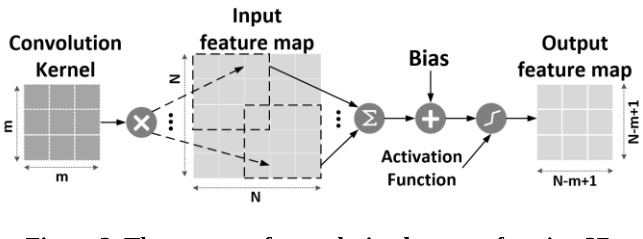

Figure 2. The process of convolution layer performing 2D convolution

The convolution layer is the core operation in the CNN architecture to extract features such as edges and lines of an input image [13]. Figure 2 shows the process of a convolution layer. First, a convolution kernel is convolved with an input feature map as it moves over the input image with a stride. Next, weighted results are added with a bias. Finally, an activation function such as sigmoid, tanh or ReLU is applied to the accumulated values to determine the output feature map. If the convolution of an $m\times m$ convolution kernel and an $N\times N$ input feature map is conducted with the stride of 1, an $(N\text{-}m+1)\times(N\text{-}m+1)$ output feature map is generated. The process in the overall convolution layer may be summarized as (1).

$$O_t^{(x,y)} = f\left(\sum_{i=0}^{m-1}\sum_{j=0}^{m-1} W_t^{(i,j)} \cdot I^{(x+i,y+j)} + bias\right) \qquad (1)$$

where a two-dimensional coordinate (x, y) indicates the position in the output feature map. I, O, and W indicate an input feature map, an output feature map, and a convolution kernel, respectively. Indices t, i, and j denote the convolution kernel type, the width index, and the height index, respectively. Finally, f denotes the activation function. For instance, if a 32×32 input feature map and 6 types of 5×5 convolution kernels are used in convolution, 6 types of 28×28 output feature maps are generated.

2.2 Sub-sampling layer

The sub-sampling layer reduces the size of feature maps from the previous convolution layer. The sub-sampling layer reduces computational complexity through image down-sampling while preserving features on images. Max-pooling, the method of choosing the largest value, is commonly used to deliver strong signals to the next layer. In addition, this layer is effective in preventing a problem known as overfitting.

2.3 Fully-connected layer

Following the feature extraction through convolution layers and sub-sampling layers, global features that can represent an input image are obtained. These features are given as inputs to the fully-connected layer. Then, the fully-connected layer classifies an input image into a category of the highest probability based on the results of the feature extraction.

3. SIMD IMPLEMENTATION OF LENET-5

3.1 Single Instruction Multiple Data (SIMD)

The SIMD unit is the implementation of the instruction set that operates on 1-D arrays called vectors. Vectors contain multiple data elements and the number of data elements per vector is typically called as vector length. Each data element is assigned to a processing element unit called lane. SIMD is capable of processing multiple data elements simultaneously through vector operations. Thus, SIMD takes advantage of data parallelism and it has been widely used in signal and image processing.

Figure 3 shows the scalar operation (Figure 3 (a)) and the functionally equivalent SIMD operation of a vector with 4 lanes (Figure 3 (b)). The scalar operation is sequentially carried out with 4 iterations, but the SIMD operation gets four results by performing only one parallel operation.

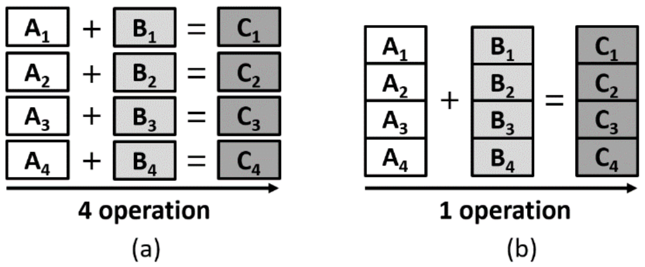

Figure 3. Comparison between scalar operation and SIMD operation

3.2 NEON

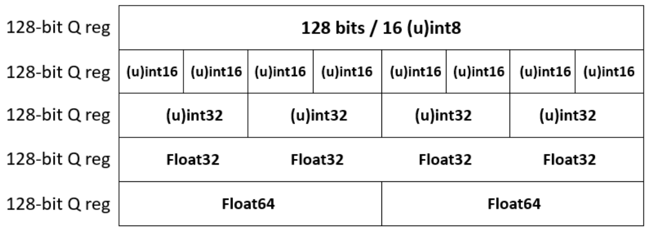

Figure 4. Data type of Q register

NEON is an advanced SIMD architecture extension for ARM processors. The NEON unit has independent pipelines and a register bank that is separate from the ARM core register bank. Registers store vectors with elements of the same data type. Data types are 8-bit, 16-bit, 32-bit, 64-bit, and 128-bit of signed/unsigned fixed-point and single precision floating-point types. The vector register of the NEON unit has 16 128-bit quad word (Q) registers (Q_0-Q_{15}) and 32 64-bit double word (D) registers (D_0-D_{31}). A Q register is composed of two consecutive D registers [11]. Figure 4 shows the number of lanes available for each data type in a Q register. For example, the 8-bit fixed-point type of data elements is used, the total of 16 lanes can be used in a vector.

3.3 Data Reshaping in Depth Directional Method

There are two types of convolution kernels in LeNet-5. *N* types of 2D convolution kernels (*H*×*W*) are used in C1, and *N* types of 3D convolution kernels (*D*×*H*×*W*) are used in C2 and C3 where *N, D, H,* and *W* denote the convolution kernel type, the kernel depth, the kernel height, and the kernel width, respectively.

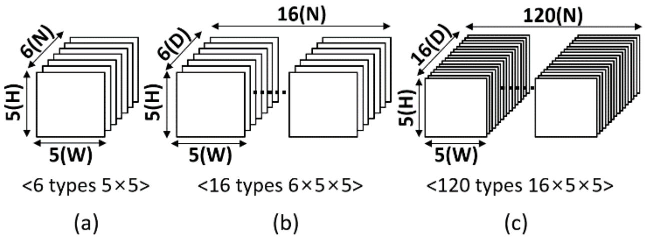

Figure 5. The shape of convolution kernels in (a) C1, (b) C2, and (c) C3

Figure 5 shows the shape of convolution kernels in LeNet-5. The sets of the convolution kernels in C1, C2, and C3 are 6 types of 5×5 kernels (Figure 5 (a)), 16 types of 6×5×5 kernels (Figure 5 (b)), and 120 types of 16×5×5 kernels (Figure 5 (c)), respectively.

As mentioned earlier, to fully utilize vector lanes in the SIMD unit, a method called Depth-Directional Method to vectorize convolution kernels in the depth direction is proposed in this paper. To vectorize the convolution kernel in the depth direction, data reshaping is performed conforming to the way of the load instruction in the NEON instruction set. When the load instruction is executed, it fetches a bundle of data of the same size as the vector length. In other words, if the vector length is larger than the number of elements in a convolution kernel row, the extra lanes are not used. We call such lane as idle lane. When loading convolution kernels into vector registers, if one row of the convolution kernel is loaded at a time in C1, only 5 lanes will be used, and the rest of the lanes will become idle lanes. On the other hand, when we reshape the data in the proposed DDM, 6 lanes will be used resulting in one less idle lane. Therefore, more lanes can be utilized by vectorizing the reshaped convolution kernel than the original convolution kernel.

Figure 5 shows the shape of convolution kernels in LeNet-5. In 2D convolution kernels, the kernel vector size will be *W*, and therefore, the number of kernel vectors is *N*×*H*. Also, in 3D convolution kernels, the kernel vector size will be *W*, and the number of kernel vectors will be *N*×*D*×*H*.

After the kernel is reshaped, the kernel vector size will be *N*, and the number of kernel vectors will become *W*×*H* in 2D convolution kernels, and the kernel vector size will be *D* and the number kernel vectors will be *N*×*W*×*H* in 3D convolution kernels.

Figure 6 shows the reshaped convolution kernel of each convolution layer. In the 2D convolution kernel of C1 (Figure 6 (a)), *N* is regarded as the depth. Therefore, the convolution kernels in the shape of 6×5×5 are reshaped to those of 5×5×6. In the 3D convolution kernels of C2 and C3, the convolution kernels in the shape of 16 types of 6×5×5 and 120 types of 16×5×5 is reshaped to those of 16 types of 5×5×6 and those of 120 types 5×5×16 as shown in Figure 6 (b) and Figure 6 (c), respectively.

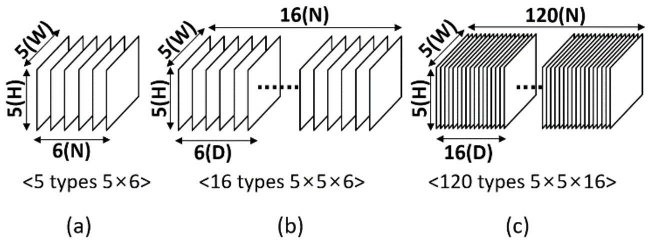

Figure 6. Reshaped convolution kernels in (a) C1, (b) C2, and (c) C3

In the proposed implementation, the data type of weights and input features is 16-bit fixed-point representation, and therefore, the number of lanes will be 8 because each Q vector register has 128 bits as shown in Figure 4. In C1, the number of lanes used per vector is increased from 5 to 6 in DDM. In C2, the number of the convolution kernel types is 16, and thus, the number of idle lanes will increase by 16 times compared to C1. In C3, as the number of lanes used per vector is 8, in DDM, all the lanes are used without any idle lanes.

Figure 7 shows how lanes are utilized in the conventional method and DDM. In C1, the number of idle lanes per vector is 3 in the conventional method, and 2 in DDM. Correspondingly, the number of vectors will be 30 and 25 vectors in the conventional method and DDM, respectively. Therefore, the total number of idle lanes is 90 and 50 in the conventional method and DDM, respectively. Figure 8 shows how lanes are utilized in the conventional method and DDM for C3. In the reshaped kernel, the vector size is 16, and therefore, as shown in Figure 8 (b), two vector registers are used per a row. Therefore, the number of vectors will be N×W×H×2, and all lanes are fully utilized without any idle lanes. Table 1 summarizes the comparison of the utilization of SIMD lanes in each convolution layer.

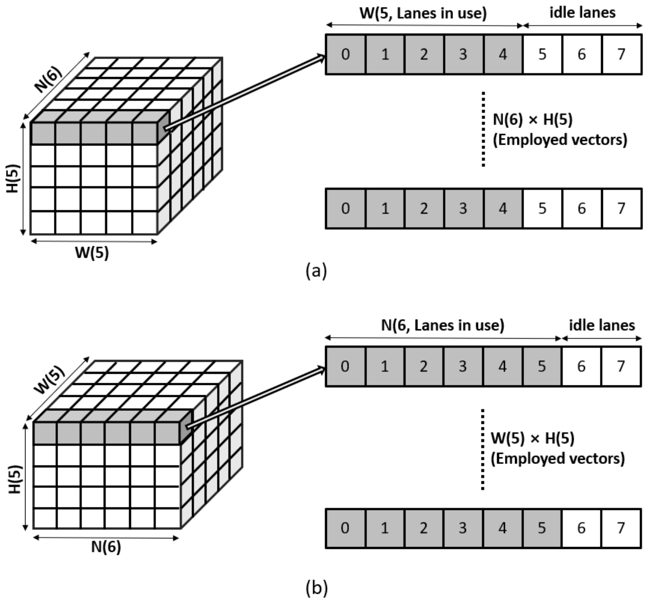

Figure 7. Utilization of SIMD lanes for C1 in (a) conventional method and (b) depth directional method

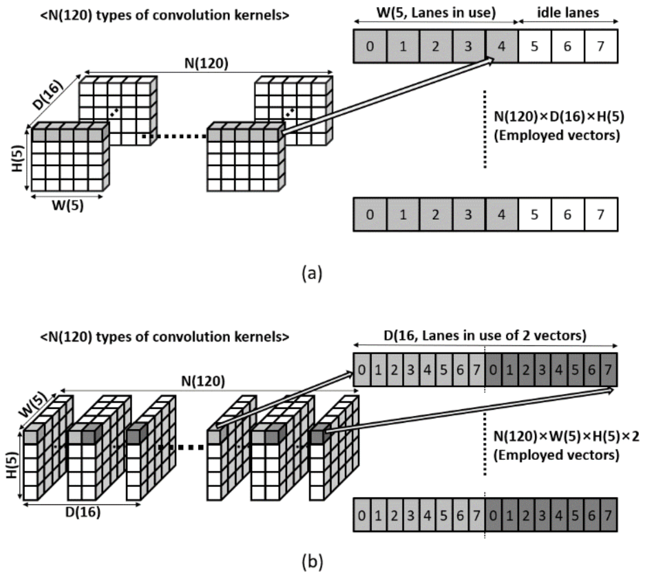

Figure 8. Utilization of SIMD lanes for C3 in (a) conventional method and (b) depth directional method

Table 1. The utilization of SIMD lanes comparison in conventional and depth directional method

	Conventional Method		**Depth Directional Method**	
Layer Type	idle lanes per vector	the number of vectors	idle lanes per vector	the number of vectors
C1	3	30	2	25
C2	3	16×30	2	16×25
C3	3	120×80	0	120×25×2

4. EXPERIMENTS

4.1 Experimental Setup

To evaluate the performance of the proposed SIMD method, we use an embedded platform named Raspberry Pi 3 MODEL B which is equipped with a quad ARM Cortex-A53 core processor with a maximum clock speed of 1.2GHz and a 1GB LPDDR2 RAM [14]. A NEON unit for advanced SIMD processing is integrated in each Cortex-A53 core.

We compare the proposed DDM with other implementations in terms of execution time and power efficiency. First, three versions of implementations have been designed for single-core evaluation: a basic C code (BASIC), a NEON SIMD code with the conventional method (NEON), a NEON SIMD code with DDM ($NEON_D$). In addition, three different implementations are designed for multi-core evaluation. The multi-core implementations are parallelized by OpenMP. OpenMP is an API for shared-memory parallel programming [15]. Compared multi-core implementations are a C code parallelized with OpenMP (OMP), a NEON SIMD code with the conventional method and OpenMP (OMP_c), a NEON SIMD code with DDM and OpenMP (OMP_D).

All implementations are experimented with the MNIST dataset. MNIST is a widely used handwritten dataset containing 28×28 pixel images representing a single digit with the class labels from 0 to 9. The dataset includes 60,000 training samples and 10,000 testing samples [16]. In experiments, all weights and input features have the 16-bit fixed-point type. ReLU is used for the activation function. The execution time and the energy dissipation of the feature extraction during inference of 10,000 images have been measured. Energy dissipation of each implementation is measured by a pluggable power meter.

4.2 Evaluation for Single-core Implementation

Table 2 shows the comparison of execution time and energy dissipation between the proposed DDM ($NEON_D$) and other implementations. $NEON_D$ shows the best execution time compared to other implementations. $NEON_D$ shows a speedup of up to 3.45 in C1 compared to BASIC. Table 3 shows relative performance and power dissipation ratios. For C1, $NEON_D$ is 41% faster than NEON, and for C2, 21% of speed improvement is achieved. For C3, $NEON_D$ is 50% faster than NEON.

Table 2. Execution time and energy dissipation comparison for single-core

	BASIC		NEON		NEON$_D$	
Layer type	Time (ms)	Energy (J)	Time (ms)	Energy (J)	Time (ms)	Energy (J)
C1	4380	110.39	1701	24.50	1107	11.96
C2	8549	307.78	3071	55.29	2501	36.01
C3	1647	11.86	673	4.85	467	1.69

Table 3. Execution time and energy dissipation ratio for single-core

	BASIC / NEON$_D$		NEON / NEON$_D$	
Layer type	Speed up	Energy Dissipation	Speed up	Energy Dissipation
C1	3.96	9.23	1.54	2.05
C2	3.42	8.55	1.23	1.54
C3	3.53	7.04	1.44	2.88

4.3 Evaluation for Multi-core Implementation

All multi-core implementations are parallelized with 4 threads. The iteration of each loop is divided into 4 groups and one group is assigned to each thread. Table 4 shows the comparison of execution time and energy dissipation with three implementations: OMP, OMP$_C$, and OMP$_D$. The proposed implementation, OMP$_D$, shows the best performance and the lowest energy dissipation than the others. OMP$_D$ shows a speedup of up to 3.01 compared to OMP as shown in Table 5.

These results confirm that the proposed depth directional method is an effective technique to improve not only the execution time but also the energy efficiency in both single and multi-core implementations.

Table 4. Execution time and energy dissipation comparison for multi-core

	OMP		OMP$_C$		OMP$_D$	
Layer type	Time (ms)	Energy (J)	Time (ms)	Energy (J)	Time (ms)	Energy (J)
C1	1929	41.69	1261	18.16	474	5.12
C2	3707	93.44	803	8.67	652	6.35
C3	982	7.07	394	1.92	171	0.61

Table 5. Execution time and energy dissipation ratio for multi-core

	OMP / OMP$_D$		OMP$_C$ / OMP$_D$	
Layer type	Speed up	Energy Dissipation	Speed up	Energy Dissipation
C1	4.07	8.14	2.66	3.55
C2	5.69	14.72	1.23	1.37
C3	5.74	11.52	2.30	3.12

5. CONCLUSIONS

Resource-constrained embedded platforms cannot afford to expensive accelerators. Therefore, it is important to accelerate CNN by only the CPU efficiently. In this paper, we propose a method to accelerate CNN by using the Single Instruction Multiple Data (SIMD) unit. To fully utilize the processing capability of the SIMD unit, we proposed a method called *Depth Directional Method*. Experimental results showed that the proposed method is superior to other conventional methods in all convolution layers. The multi-core implementation with the proposed method achieved a speedup of up to 13.11 in execution time and an improvement of up to 48.5 times in energy dissipation compared to the other conventional single-core implementations.

6. ACKNOWLEDGMENTS

This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education(NRF2015R1D1A1A09061079).

7. REFERENCES

[1] Simard, P. Y., Steinkraus, D., and Platt, J. C. 2003. Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis. In *Proceedings of the Seventh International Conference on Document Analysis and Recognition*, 2 (ICDAR '03). IEEE Computer Society, Washington, DC, USA, 958-. DOI=http://dx.doi.org/10.1109/ICDAR.2003.1227801.

[2] Abdel-Hamid, O., Mohamed, A., Jiang, H., Deng, L., Penn, G., and Yu, D. 2014. Convolutional Neural Networks for Speech Recognition. *IEEE/ACM Trans. Audio, Speech and Lang. Proc.* 22, 10 (Oct. 2014), 1533-1545. DOI=http://dx.doi.org/10.1109/TASLP.2014.2339736.

[3] Collobert, R. and Weston, J. 2008. A unified architecture for natural language processing: deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning* (ICML '08). ACM, New York, NY, USA, 160-167. DOI=http://dx.doi.org/10.1145/1390156.1390177.

[4] Liu, S. and Deng, W. 2015. Very deep convolutional neural network based image classification using small training sample size. In *Proceedings of 3rd IAPR Asian Conference on Pattern Recognition (ACPR)*, 730-734. DOI=http://dx.doi.org/10.1109/ACPR.2015.7486599.

[5] He, K., Zhang, X., Ren, S., and Sun, J. 2016. Deep Residual Learning for Image Recognition, In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (CVPR), 770-778. DOI=http://dx.doi.org/10.1109/CVPR.2016.90.

[6] Lomont, C. 2011. Introduction to Intel Advanced Vector Extensions. Intel White Paper.

[7] ARM. *Architecture support for NEON and VFP*. http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0204j/CJAJBFBF.html/

[8] Michael, J. F. 1966. Very high-speed computing systems. In *Proceedings of the IEEE*. 54, 1901-1909. DOI=http://dx.doi.org/10.1109/PROC.1966.5273.

[9] Siegel, H. J., Siegel, L. J., Kemmerer, F. C., PT Jr, M., HE Jr, S., and Smith, S. D. 1981. PASM: A partitionable SIMD/MIMD system for image processing and pattern recognition. *IEEE Transactions on computers*, 30, 12 (Dec. 1981), 934-947. DOI=http://dx.doi.org/10.1109/TC.1981.1675732.

[10] Lai, L., Suda, N., and Chandra, V. 2018. CMSIS-NN: Efficient Neural Network Kernels for Arm Cortex-M CPUs. arXiv preprint arXiv:1801.06601.

[11] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. 1998. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, 86, 11 (Nov 1998), 2278-2324. DOI=http://dx.doi.org/10.1109/5.726791.

[12] Krizhevsky, A., Sutskever, I., and Hinton, G. E. 2017. ImageNet classification with deep convolutional neural networks. Commun. ACM 60, 6 (May 2017), 84-90. DOI: https://doi.org/10.1145/3065386.

[13] Chen, L. C., Barron, J. T., Papandreou, G., Murphy, K., & Yuille, A. L. 2016. Semantic image segmentation with task-specific edge detection using cnns and a discriminatively trained domain transform. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (CVPR), 4545-4554. DOI=http://dx.doi.org/10.1109/CVPR.2016.492.

[14] Raspberry PI Foundation. *RASPBERRY PI 3 MODEL B.* https://www.raspberrypi.org/products/raspberry-pi-3-model-b 2016/

[15] OpenMP. *OpenMP Specifications*. http://www.openmp.org/specifications.

[16] LeCun, Y., Cortes, C., Burges, C. J. 2010. *MNIST handwritten digit database*. AT&T Labs. http://yann.lecun.com/exdb/mnist.