Direct Conversion: Accelerating Convolutional Neural Networks Utilizing Sparse Input Activation

Won-Hyuk Lee, Si-Dong Roh, Sangki Park, and Ki-Seok Chung*
Department of Electronics and Computer Engineering
Hanyang University
Seoul, Korea
Email: {louislee111, sdroh1027, skpark1101, kchung}@hanyang.ac.kr

***Abstract*—The amount of computation and the number of parameters of neural networks are increasing rapidly as the depth of convolutional neural networks (CNNs) is increasing. Therefore, it is very crucial to reduce both the amount of computation and that of memory usage. The pruning method, which compresses a neural network, has been actively studied. Depending on the layer characteristics, the sparsity level of each layer varies significantly after the pruning is conducted. If weights are sparse, most results of convolution operations will be zeroes. Although several studies have proposed methods to utilize the weight sparsity to avoid carrying out meaningless operations, those studies lack consideration that input activations may also have a high sparsity level. The Rectified Linear Unit (ReLU) function is one of the most popular activation functions because it is simple and yet pretty effective. Due to properties of the ReLU function, it is often observed that the input activation sparsity level is high (up to 85%). Therefore, it is important to consider both the input activation sparsity and the weight one to accelerate CNN to minimize carrying out meaningless computation. In this paper, we propose a new acceleration method called *Direct Conversion* that considers the weight sparsity under the sparse input activation condition. The Direct Conversion method converts a 3D input tensor directly into a compressed format. This method selectively applies one of two different methods: a method called *image to Compressed Sparse Row* (im2CSR) when input activations are sparse and weights are dense; the other method called *image to Compressed Sparse Overlapped Activations* (im2CSOA) when both input activations and weights are sparse. Our experimental results show that Direct Conversion improves the inference speed up to 2.82× compared to the conventional method.**

***Index Terms*—convolutional neural network, sparsity-aware acceleration, embedded system**

I. Introduction

Convolutional neural networks (CNNs) have achieved great success in various fields such as image classification [1], image detection [2], and semantic segmentation [3]. Because modern CNN architectures are getting deeper to achieve a higher accuracy [4], computational cost, memory cost, and inference latency are increasing rapidly. To improve the inference speed, lowering methods were introduced. The lowering method typically includes a transformation of a 3-dimension (3D) input tensor to a 2D input matrix. Since the kernel matrix is stored in a 2D matrix format, the lowering step is not necessary. Therefore, a 2D input matrix and a 2D kernel matrix can be multiplied with a conventional general matrix multiplication (GEMM). Commonly, to effectively conduct GEMM, transformations such as *image to column* (im2col) [5] and *image to row* (im2row) [6] are preprocessed. However, many pruning methods targeted for resource-limited computing systems tend to convert most of the weights or the activations to zero [7], and therefore, lots of meaningless computations are carried out in conventional GEMM operations. Moreover, lowering transformations suffer from significant latency and memory overhead to generate temporary matrices.

To avoid these problems, a method called *direct sparse convolution* that accelerates effectively a pruned-CNN with sparse weights was introduced [8], [9]. In this method, instead of the lowering transformation, a sparse kernel matrix is represented as a format called *Compressed Sparse Row* (CSR) [10]. CSR is a compression format that extracts only nonzero values from a sparse matrix. Since the CSR format maintains only the non-zero values to carry out only the meaningful computations, the sparse convolution with CSR has no loss of accuracy. Nevertheless, this method takes only the weight sparsity into account and does not consider the input activation sparsity. Thus, this may be inefficient for the layers where input activations are sparse.

In this paper, we propose a new acceleration method called *Direct Conversion* that considers the weight sparsity under the sparse input activation condition. To maximize the effectiveness of the input tensor processing, the following issues need to be addressed. First, it should be noted that the overhead to lower the dimension of a 3D input tensor to a 2D input matrix form is significant. Especially, when the input tensor is sparse, the lowering overhead will become too wasteful [11]. Second, since the input activation sparsity and the weight sparsity are mutually independent, it is possible that weights can be either dense or sparse regardless of whether input activations are sparse or not. To take these two issues into account, the proposed Direct Conversion method converts a 3D input tensor directly into a compression format such as CSR. And

TABLE I
Applied methods based on the sparsity level of weight and input activation

Activation \ Weight	Dense	Sparse
Dense	im2row	Direct Sparse Convolution
Sparse	**im2CSR**	**im2CSOA**

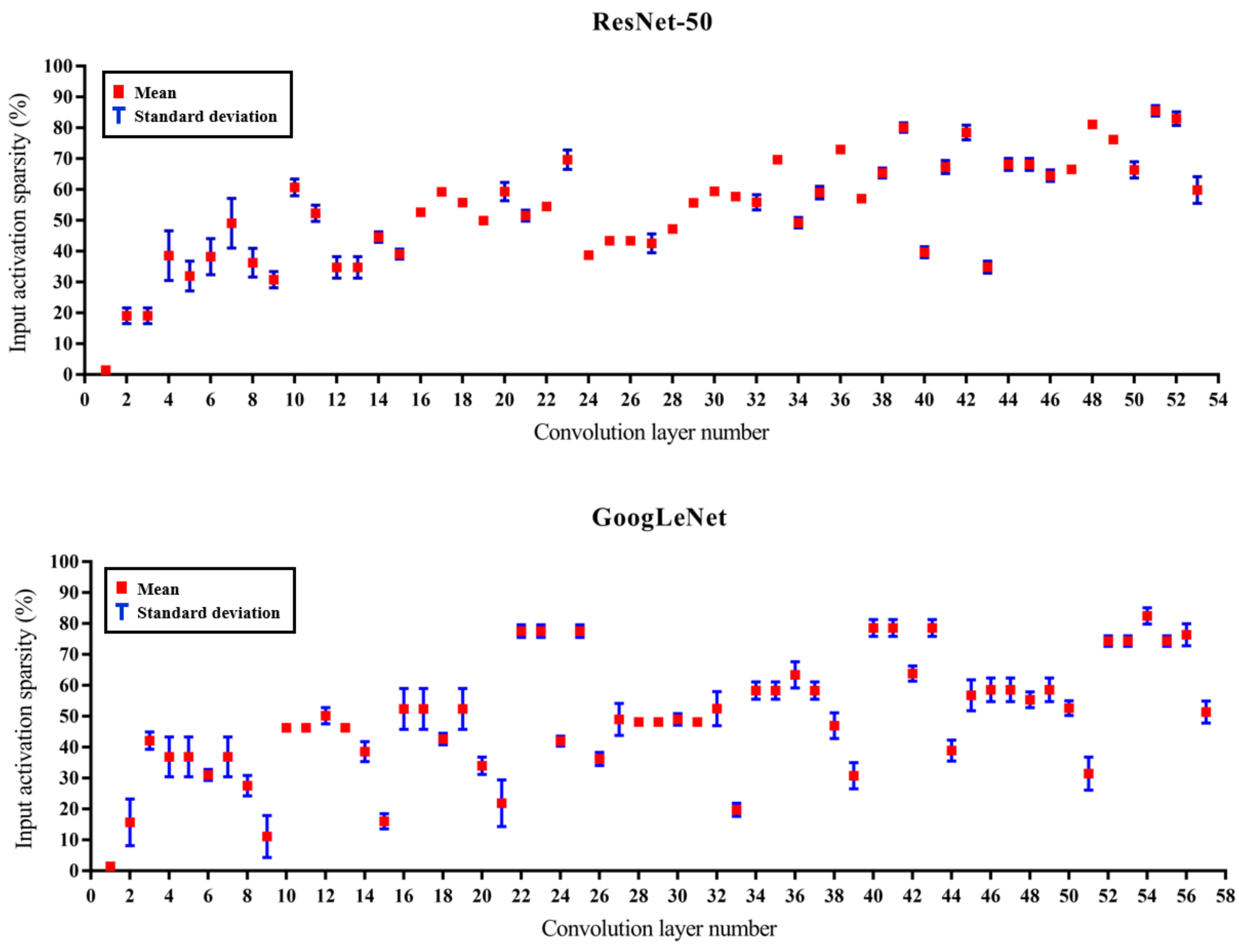


Fig. 1. Analysis of the population mean and the population standard deviation of input activation sparsity

one of two different methods is selectively applied: a method called *image to Compressed Sparse Row* (im2CSR) when the input activations are sparse and the weights are dense; the other method called *image to Compressed Sparse Overlapped Activations* (im2CSOA) when both the input activations and the weights are sparse. In other words, when only the input activation is sparse, a method called *Sparse Matrix Dense Matrix Multiplication* (SpMDM) is applied, and when both are sparse, a method called *Sparse Matrix Sparse Matrix Multiplication* (SpMSpM) is selected as listed in Table I.

II. Estimation of Activation and Weight Sparsity

In order to apply an appropriate method based on the sparsity level, it is necessary to accurately estimate the sparsity level of both input activations and weights for each layer. The sparsity level of the weights and that of the input activations are dependent on the applied weight pruning method and the applied activation function, respectively. That is, the weights of which absolute values are close to zero will be set to zero through the weight pruning, and the activation function like rectified linear unit (ReLU) generates the zero value when the result of an operation is a negative value [7], [12]. Especially, ReLU is one of the popular activation functions for CNN because it is simple, yet effective. When ReLU is used as the activation function, the input activation sparsity reaches up to 85% in many convolution layers.

Estimation of the weight sparsity level for a trained network is rather straightforward because all the weights will be fixed after training is complete. On the other hand, the sparsity level of the input activation may vary from layer to layer. However, from running inference on CNNs with lots of images, it is observed that sparsity levels of the input activation reveal some consistent patterns regardless of the input images.

Fig. 1 shows the sparsity level of the input activation for each convolution layer when the ImageNet training dataset [13] with 1,281,167 images was used to run inference on ResNet-50 [14] and GoogLeNet [4]. The red and blue symbols denote the population mean (PM) and the population standard deviation (PSD), respectively. It shows both CNNs have small PSD values and have a similar pattern that the deeper layer it reaches, the bigger the PM gets, and the smaller the PSD gets. Small PSDs imply that the input activation sparsity stays with similar levels regardless of the input images. Thus, the sparsity level of the input activation of each layer can be analyzed in advance. In this paper, the layers with the sparsity level of 65% or higher are categorized as a sparse layer.

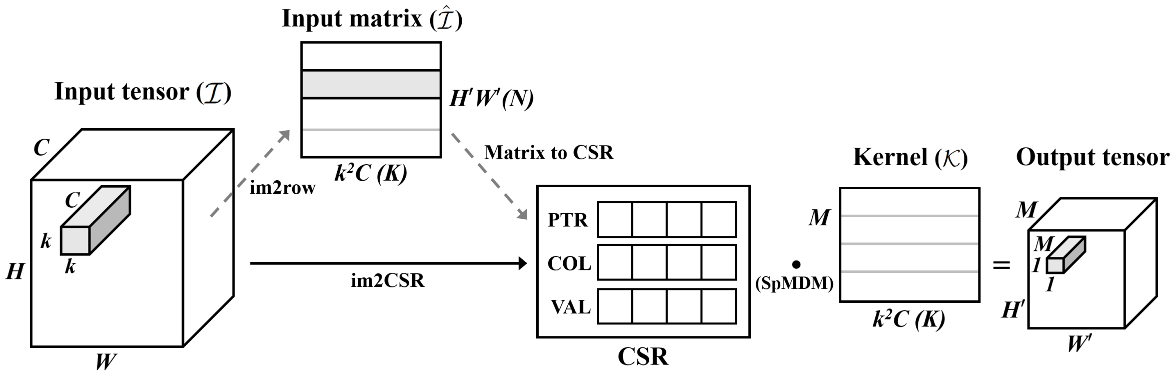


Fig. 2. Process of image to Compressed Sparse Row (im2CSR)

Algorithm 1 Image to Compressed Sparse Row

```
    Input: In (Input tensor), Weight (Weight matrix)
    Output: Out (Output tensor)
    Compression format: B (CSR format)
 1: B.PTR[0] = 0
 2: for $(h, w) \leftarrow (0, 0)$ to $(H, W)$ do
 3:     for $(c, x, y) \leftarrow (0, 0, 0)$ to $(C, k, k)$ do
 4:         if In$[c][h + x][w + y] \neq 0$ then
 5:             B.VAL$[B_{NNZ}]$ = In$[c][h + x][w + y]$
 6:             B.COL$[B_{NNZ}] = (c \times k + x) \times k + y$
 7:             $B_{NNZ} = B_{NNZ} + 1$
 8:         end if
 9:     end for
10:     B.PTR$[h \times H + w + 1] = B_{NNZ}$
11: end for
12: for $(i, j) \leftarrow (0, 0)$ to $(M, N)$ do
13:     for $off \leftarrow$B.PTR$[j]$ to B.PTR$[j+1]$-B.PTR$[j]$ do
14:         $sum$ += B.VAL$[off]$ × Weight$[i]$[B.COL$[off]$]
15:     end for
16:     Out$[i][j] = sum$
17: end for
```

III. Direct Conversion

A. Overview

In this paper, we propose a new acceleration method called *Direct Conversion* that considers the weight sparsity under the sparse input activation condition. Direct Conversion focuses on achieving two goals: minimizing the overhead due to lowering tensors and accelerating the CNN based on the sparsity level.

First, to minimize the overhead of lowering, the input activation is not transformed into a matrix form. Typically, a sparse matrix is stored as a compressed format such as CSR. The CSR format consists of three arrays: PTR that saves the accumulated count of nonzero elements, COL that saves the column index of nonzero elements on each row, and VAL that saves the nonzero values. To create a CSR using im2row, lowering $k \times k \times C$ receptive fields in a 3D input tensor $\mathcal{I} \in \mathbb{R}^{C \times H \times W}$ to rows of a 2D input matrix $\hat{\mathcal{I}} \in \mathbb{R}^{N \times K}$ should be preceded. C, H, and W denote the channel, the height, and the width of the input, respectively while k is the height and the width of the kernel, and N and K are the height×width of the output and $k \times k \times C$, respectively. After the preprocessing, converting the input matrix $\hat{\mathcal{I}}$ to a CSR will follow as illustrated using dotted arrows in Fig. 2. This two-step conversion process through the lowering step incurs significant delay and memory overhead. Second, because the input activation sparsity and the weight sparsity are mutually independent, it is desirable to apply different acceleration methods depending on the weight sparsity level. In Direct Conversion, methods called im2CSR and im2CSOA are selectively employed to convert an input tensor directly to either a CSR or a CSOA as shown using a solid arrow in Fig. 2 and Fig. 4, respectively. As shown in Table I, the im2CSR method is applied when the weights are dense, and the im2CSOA method is applied when the weights are sparse.

B. Image to Compressed Sparse Row

When a layer has sparse input activations and dense weights, only the sparse input tensor $\mathcal{I}$ is converted directly into a compression format without going through the lowering step. As a Direct Conversion method, the im2CSR method that converts the sparse input tensor $\mathcal{I}$ into a CSR format directly is proposed. The im2CSR algorithm for converting from a sparse input tensor to a CSR is described in from line 2 through line 11 in Algorithm 1. Once a CSR representation is generated, the CSR will be multiplied by a dense kernel matrix $\mathcal{K} \in \mathbb{R}^{M \times K}$ where M is the channel of the output using a method called *Sparse Matrix Dense Matrix Multiplication* (SpMDM) [15] to generate an output tensor. SpMDM is a method to multiply a compression format of a sparse matrix by a dense matrix. The inner product based SpMDM steps are described in from line 12 to line 17 in Algorithm 1, where B is a CSR format and B_{NNZ} represents the number of nonzero elements (NNZ) in B. To execute the inner product based SpMDM, the NNZ information of the current row should be available. To obtain this information, the accumulated count of the current NNZ value should be subtracted from the accumulated count of the next NNZ value. The inner product based SpMDM described in line 14 is repeated through the obtained NNZ. As a result, im2CSR reduces latency and memory overhead and

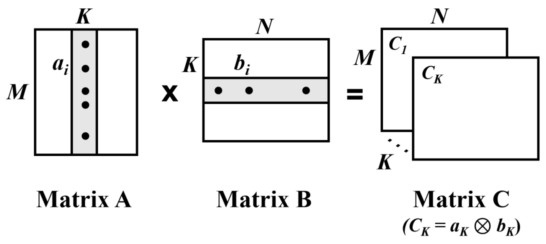


Fig. 3. Example of the outer product method of sparse matrices A and B

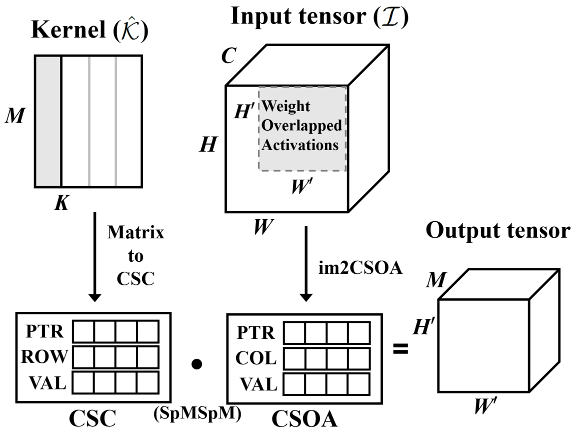
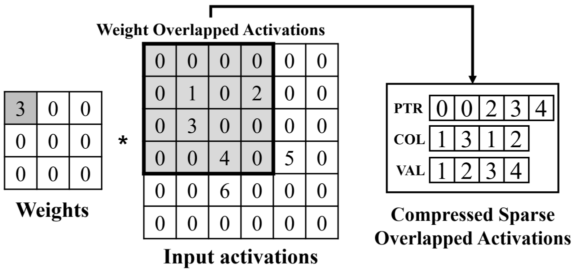


Fig. 4. Process of image to Compressed Sparse Overlapped Activations (im2CSOA)


Weight Overlapped Activations
Weights
Input activations
PTR
COL
VAL
Compressed Sparse Overlapped Activations
Fig. 5. Example of Compressed Sparse Overlapped Activations (CSOA)

accelerates efficiently in layers with sparse input activations and dense weights.

C. Image to Compressed Sparse Overlapped Activations

When both input matrices are sparse, a method called *Sparse Matrix Sparse Matrix Multiplication* (SpMSpM) [16] is employed. However, the inner product based SpMSpM is inefficient because the inner product multiplication should be performed selectively on matched indices of nonzero elements. Therefore, checking whether row and column indices are matched or not should be carried out for each operation. However, if the outer product method is used in SpMSpM, such index-matching step can be eliminated. Furthermore, reusing nonzero elements can be maximized leading to minimized loads of columns and rows [16]. The outer product method gets the final result through merging results of multiplication of pairs of columns of the first input matrix and rows of the second input matrix as illustrated in Fig. 3.

Algorithm 2 Image to Compressed Sparse Overlapped Activations

```
    Input: In (Input tensor), Weight (Weight matrix)
    Output: Out (Output tensor)
    Compression format: A (CSC format), B (CSOA format)
 1: A.PTR[0] = 0, B.PTR[0] = 0
 2: for $j \leftarrow 0$ to $K$ do
 3:     for $i \leftarrow 0$ to $M$ do
 4:         if Weight$[i][j] \neq 0$ then
 5:             A.VAL$[A_{NNZ}]$ = Weight$[i][j]$
 6:             A.ROW$[A_{NNZ}] = i$
 7:             $A_{NNZ} = A_{NNZ} + 1$
 8:         end if
 9:     end for
10:     A.PTR$[j+1] = A_{NNZ}$
11: end for
12: for $(c, x, y) \leftarrow (0, 0, 0)$ to $(C, k, k)$ do
13:     for $(h, w) \leftarrow (0, 0)$ to $(H, W)$ do
14:         if In$[c][h+x][w+y] \neq 0$ then
15:             B.VAL$[B_{NNZ}]$ = In$[c][h+x][w+y]$
16:             B.COL$[B_{NNZ}] = h \times W + w$
17:             $B_{NNZ} = B_{NNZ} + 1$
18:         end if
19:     end for
20:     B.PTR$[(c \times k + x) \times k + y + 1] = B_{NNZ}$
21: end for
22: for $i \leftarrow 0$ to $K$ do
23:     for $A_{off} \leftarrow$ A.PTR$[i]$ to A.PTR$[i+1]$ - A.PTR$[i]$ do
24:         for $B_{off} \leftarrow$B.PTR$[i]$ to B.PTR$[i+1]$-B.PTR$[i]$ do
25:             Out[A.ROW$[A_{off}]$][B.COL$[B_{off}]$] +=
                    A.VAL$[A_{off}]\times$ B.VAL$[B_{off}]$
26:         end for
27:     end for
28: end for
```

Based on the outer product based SpMSpM, we propose a method called *image to Compressed Sparse Overlapped Activations* (im2CSOA). In im2CSOA, a sparse kernel matrix $\hat{\mathcal{K}} \in \mathbb{R}^{M \times K}$ is converted to a *Compressed Sparse Column* (CSC) format and the sparse input tensor $\mathcal{I}$ is directly converted to CSOA without the lowering step as shown in Fig. 4. While CSR stores only the nonzero elements in each row in contiguous memory locations, CSC is similar to CSR with the exception that each column is compressed. In CSOA, the compression is carried out with a unit called *weight overlapped activations*. The weight overlapped activations are defined as the set of input activation elements of which index in the sliding window matches that of a certain weight element when the window moves over the input activation matrix. For instance, in Fig. 5, the index of weight value, 3 in the weight matrix is (0,0). Then, the weight overlapped activations for the weight will be the grey-colored submatrix because the submatrix corresponds to the set of elements that will have

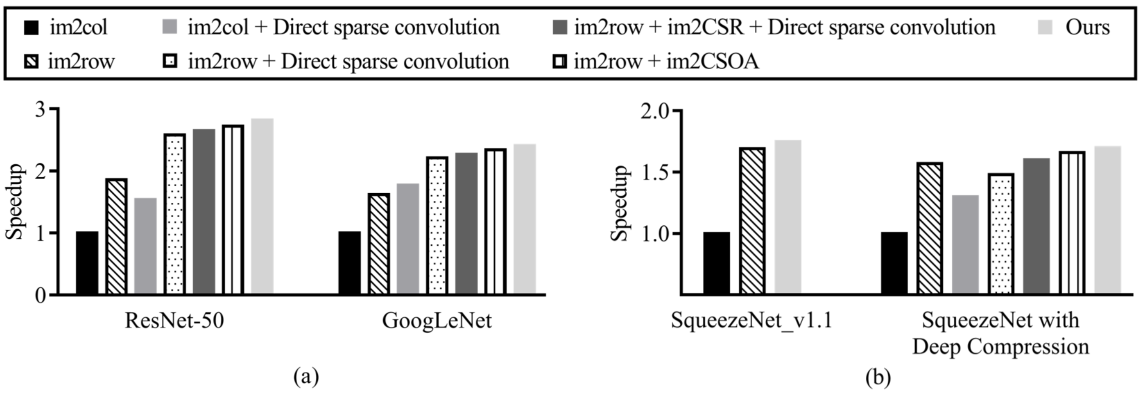


Fig. 6. Inference time speedup of conventional methods and Direct Conversion over im2col for large-scale networks (a) and small-scale networks (b)

the same index (i.e. (0,0)) as the window slides over the input activation matrix.

The column of kernel matrix $\hat{\mathcal{K}}$ and the weight overlapped activations of the input tensor matrix in Fig. 4 correspond to the column of Matrix A and the row of Matrix B in Fig. 3, respectively. Therefore, im2CSOA converts all columns of kernel matrix $\hat{\mathcal{K}}$ to CSC and all the weight overlapped activations of the input tensor to CSOA. After the conversion, the outer product based SpMSpM is conducted to generate an output tensor as illustrated in Fig. 3. Algorithm 2 shows pseudo-codes to describe the implementation of the im2CSOA method. In summary, by combining the Direct Conversion method with the outer product based SpMSpM, the proposed method achieves significant improvement over the conventional methods.

IV. Experiment Results

A. Experimental Setup

We evaluated the performance of Direct Conversion on a 1.43GHz ARM Cortex-A57 quad-core processor using the ImageNet validation dataset [13]. The proposed Direct Conversion method was implemented as a Caffe deep learning framework [17]. As aforementioned, the applied methods based on the sparsity level of the weight and that of the input activation are summarized in Table I. Our implementation utilizes a 4-way ARM advanced single instruction multiple data (SIMD) unit for operations with 32-bit floating-point numbers [18] and exploits thread-level parallelism with OpenMP [19]. We evaluated the performance of compared methods on two different network categories: a large-scale network and a small-scale one. First, ResNet-50 [14] and GoogLeNet [4] were selected as large-scale networks. We used trained and pruned networks of ResNet-50 and GoogLeNet, which are available in the Intel SkimCaffe repository [20]. Second, SqueezeNet_v1.1 and SqueezeNet with Deep Compression [21], [22] were chosen as small-scale networks that are suitable for resource-constrained embedded systems.

B. Large-scale Networks

Fig. 6 (a) shows the normalized inference time speedup of Direct Conversion compared to conventional methods for the ResNet-50 and the GoogLeNet. The im2col method is used as the baseline for comparison. The inference time speedup of im2row over im2col is about 1.86×. The *im2col + direct sparse convolution* method uses im2col when weights are dense, and direct sparse convolution when the weights are sparse [8], [9]. This method shows a slower inference time compared to im2row in the ResNet-50 mainly due to the underperforming im2col. Consequently, the *im2row + direct sparse convolution* method shows a better performance than im2row when the weight is dense. The *im2row + im2CSR + direct sparse convolution* method achieves a speedup of 2.65× over im2col mainly because either im2row or im2CSR is selectively applied based on the input activation sparsity level of each layer. The *im2row + im2CSOA* method shows a speedup of 2.72× over im2col. In this method, when the weights are dense and the input activations are sparse, im2CSOA is employed instead of direct sparse convolution because im2CSOA is faster than direct sparse convolution regardless of the input activation sparsity. Lastly, our proposed method that selectively applies im2row, im2CSR, and im2CSOA methods shows the best inference time, and achieves a speedup of 2.82× over the im2col method.

C. Small-scale Networks

SqueezeNet is a small-scale network that minimizes the amount of computation and the number of parameters so that it should be suitable for carrying out inference on embedded systems. SqueezeNet_v1.1 was chosen because it has the same classification accuracy but 2.4× fewer operations than SqueezeNet_v1.0. We did not apply im2CSOA and direct sparse convolution in this experiment because this network is not pruned. That is, most weights are quite dense. SqueezeNet with Deep Compression is a network where SqueezeNet_v1.0 is pruned with Deep Compression [22] as the pruning method, and thus, it has sparse weights. Therefore, for this network, we have applied the methods for sparse weights. This experiment compares the inference time speedup between Direct Conversion and various conventional methods.

Fig. 6 (b) shows the normalized inference time as the im2col method is used as the baseline. For SqueezeNet_v1.1, im2row

TABLE II
INFERENCE TIME SPEEDUP OF COMPARED METHODS OVER IM2COL

Method	ResNet-50	GoogLeNet	SqueezeNet_v1.1	SqueezeNet with Deep Compression
im2col	3.699s (1x)	1.536s (1x)	0.394s (1x)	0.838s (1x)
im2row	1.985s (1.86x)	0.95s (1.62x)	0.233s (1.69x)	0.534s (1.57x)
im2col + Direct sparse convolution	2.406s (1.54x)	0.867s (1.77x)	-	0.645s (1.3x)
im2row + Direct sparse convolution	1.434s (2.58x)	0.694s (2.21x)	-	0.564s (1.48x)
im2row + im2CSR + Direct sparse convolution	1.396s (2.65x)	0.675s (2.27x)	-	0.524s (1.6x)
im2row + im2CSOA	1.359s (2.72x)	0.656s (2.34x)	-	0.504s (1.66x)
Ours	**1.311s (2.82x)**	**0.637s (2.41x)**	**0.225s (1.75x)***	**0.492s (1.7x)**

*Only im2row and im2CSR is employed

is 1.69× faster than the baseline, and the proposed method is 1.75× faster. For SqueezeNet with Deep Compression, similar performance results as for large-scale networks are achieved. Our proposed method achieves 1.7× faster inference time than the baseline. To sum up, Table II summarizes all the performance results of all the compared methods for both large-scale networks and small-scale networks.

V. CONCLUSION

In this paper, we propose a new method called Direct Conversion to accelerate CNN utilizing the sparsity level of the input activation. While conventional methods only consider the sparsity level of the weights, the proposed method takes both the weight sparsity and the input activation sparsity into account. The input activation sparsity level can be as high as up to 85% when the ReLU function is used as the activation function. The proposed acceleration method includes two conversion methods: im2CSR and im2CSOA. The proposed Direct Conversion method focused on two issues: minimizing the overhead due to lowering tensors and accelerating CNNs based on the sparsity level. Our experimental results show that the proposed method achieves the best results for both large-scale networks and small-scale networks. The inference time speedups of the proposed method over im2col method are 2.82× for large-scale networks and 1.75× for small-scale networks, respectively.

ACKNOWLEDGMENT

This paper was supported by Korea Institute for Advancement of Technology(KIAT) grant funded by the Korea Government(MOTIE) (N0001883, The Competency Development Program for Industry Specialist)

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[2] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.

[3] J. Long, E. Shelhamer, and T. Darrell, “Fully convolutional networks for semantic segmentation,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3431–3440.

[4] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[5] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, “cudnn: Efficient primitives for deep learning,” *arXiv preprint arXiv:1410.0759*, 2014.

[6] A. Vedaldi and K. Lenc, “Matconvnet: Convolutional neural networks for matlab,” in *Proceedings of the 23rd ACM international conference on Multimedia*, 2015, pp. 689–692.

[7] S. Han, J. Pool, J. Tran, and W. Dally, “Learning both weights and connections for efficient neural network,” in *Advances in neural information processing systems*, 2015, pp. 1135–1143.

[8] J. Park, S. R. Li, W. Wen, H. Li, Y. Chen, and P. Dubey, “Holistic sparsecnn: Forging the trident of accuracy, speed, and size,” *arXiv preprint arXiv:1608.01409*, vol. 1, no. 2, 2016.

[9] J. Park, S. Li, W. Wen, P. T. P. Tang, H. Li, Y. Chen, and P. Dubey, “Faster cnns with direct sparse convolutions and guided pruning,” *arXiv preprint arXiv:1608.01409*, 2016.

[10] F. G. Gustavson, “Some basic techniques for solving sparse systems of linear equations,” in *Sparse matrices and their applications.* Springer, 1972, pp. 41–52.

[11] S. Hadjis, F. Abuzaid, C. Zhang, and C. Ré, “Caffe con troll: Shallow ideas to speed up deep learning,” in *Proceedings of the Fourth Workshop on Data analytics in the Cloud*, 2015, pp. 1–4.

[12] V. Nair and G. E. Hinton, “Rectified linear units improve restricted boltzmann machines,” in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807–814.

[13] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, “Imagenet large scale visual recognition challenge,” *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

[14] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[15] I. S. Duff, M. A. Heroux, and R. Pozo, “An overview of the sparse basic linear algebra subprograms: The new standard from the blas technical forum,” *ACM Transactions on Mathematical Software (TOMS)*, vol. 28, no. 2, pp. 239–267, 2002.

[16] S. Pal, J. Beaumont, D.-H. Park, A. Amarnath, S. Feng, C. Chakrabarti, H.-S. Kim, D. Blaauw, T. Mudge, and R. Dreslinski, “Outerspace: An outer product based sparse matrix multiplication accelerator,” in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA).* IEEE, 2018, pp. 724–736.

[17] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, “Caffe: Convolutional architecture for fast feature embedding,” in *Proceedings of the 22nd ACM international conference on Multimedia*, 2014, pp. 675–678.

[18] ARM, “Architecture support for neon and vfp,” 2012. [Online]. Available: https://developer.arm.com/architectures/instruction-sets/simd-isas/neon

[19] OpenMP, “Openmp specifications,” 1997. [Online]. Available: https://www.openmp.org/specifications/

[20] IntelLabs, “Skimcaffe,” 2017. [Online]. Available: https://github.com/IntelLabs/SkimCaffe

[21] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, “Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size,” *arXiv preprint arXiv:1602.07360*, 2016.

[22] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding,” *arXiv preprint arXiv:1510.00149*, 2015.