Short floating-point representation for convolutional neural network inference

Hyeong-Ju Kang

School of Computer Science and Engineering, Korea University of Technology and Education, Cheonan, Chungnam 31253, Republic of Korea
a) hjkang@koreatech.ac.kr

Abstract: Convolutional neural networks (CNNs) are being widely used in computer vision tasks, and there have been many efforts to implement CNNs in ASIC or FPGA for power-hungry environments. Instead of the previous common representation, the fixed-point representation, this letter proposes a short floating-point representation for CNNs. The short floating-point representation is based on the normal floating-point representation, but has much less width and does not have complex cases like Not-a-Number and infinity cases. The short floating-point representation, contrary to the common belief, can produce a low-complexity computation logic because the operands of the multiplier logic can be shortened by the exponent concept of the floating-point representation. The exponent can also reduce the total length to reduce the SRAM area. The experimental results show that the short floating-point representation with 8-bit total width achieves less-than-1-percentage-point degradation without the aid of retraining in the top-5 accuracy on very deep CNNs of up to 152 layers and gives more than a 60% area reduction in the ASIC implementation.

Keywords: deep learning, convolutional neural networks, number representation, neural network accelerator

Classification: Integrated circuits

References

[1] A. Krizhevsky, et al.: “ImageNet classification with deep convolutional neural networks,” Proc. Advances in Neural Inf. Process. Syst. (2012) 1097.
[2] M. Lin, et al.: “Network in network,” Proc. Int. Conf. Learning Representations (2014) http://arxiv.org/abs/1312.4400.
[3] C. Szegedy, et al.: “Going deeper with convolutions,” Proc. Comput. Vision and Pattern Recognition (2015) 1, http://arxiv.org/abs/1409.4842.
[4] K. Simonyan and A. Zisserman: “Very deep convolutional networks for large-scale image recognition,” Proc. Int. Conf. Learning Representations (2015) http://arxiv.org/abs/1409.1556.
[5] K. He, et al.: “Deep residual learning for image recognition,” Proc. Comput. Vision and Pattern Recognition (2016) 770, http://arxiv.org/abs/1512.03385.
[6] F. N. Iandola, et al.: “SqueezeNet: AlexNet-level accuracy with 50× fewer
parameters and <0.5 MB model size” (2016) http://arxiv.org/abs/1602.07360.
[7] T. Chen, et al.: “DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning,” Proc. Int. Conf. Architectural Support for Programming Languages and Operating Syst. (2014) 269 (DOI: 10.1145/2541940.2541967).
[8] Y.-H. Chen, et al.: “Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks,” Proc. Int. Symp. Comput. Architecture (2016) 367 (DOI: 10.1109/ISCA.2016.40).
[9] S. Liu, et al.: “Cambricon: An instruction set architecture for neural networks,” Proc. Int. Symp. Comput. Architecture (2016) 393 (DOI: 10.1109/ISCA.2016.42).
[10] S. Moini, et al.: “A resource-limited hardware accelerator for convolutional neural networks in embedded vision applications,” IEEE Trans. Circuits Syst. II, Exp. Briefs 64 (2017) 1217 (DOI: 10.1109/TCSII.2017.2690919).
[11] S. Gupta, et al.: “Deep learning with limited numerical precision,” Proc. Int. Conf. Mach. Learning (2015) 1737, http://arxiv.org/abs/1502.02551.
[12] D. D. Lin, et al.: “Fixed point quantization of deep convolutional networks” (2015) http://arxiv.org/abs/1511.06393.
[13] W. Sung, et al.: “Resiliency of deep neural networks under quantization” (2015) http://arxiv.org/abs/1511.06488.
[14] M. Courbariaux: “Training deep neural networks with low precision multiplications,” Proc. Int. Conf. Learning Representations (2015) http://arxiv.org/abs/1412.7024.
[15] P. Judd, et al.: “Reduced-precision strategies for bounded memory in deep neural nets” (2015) http://arxiv.org/abs/1511.05236.
[16] P. M. Gysel: “Learning multiple layers of features from tiny images,” Master’s thesis, Department of Electrical and Computer Engineering, University of California Davis (2016).
[17] M. Rastegari, et al.: “XNOR-Net: ImageNet classification using binary convolutional neural networks” (2016) http://arxiv.org/abs/1603.05279.
[18] F. Li, et al.: “Ternary weight networks” (2016) http://arxiv.org/abs/1605.04711.
[19] S. Hashemi, et al.: “Understanding the impact of precision quantization on the accuracy and energy of neural networks” (2016) http://arxiv.org/abs/1612.03940.
[20] P. Gysel, et al.: “Hardware-oriented approximation of convolutional neural networks,” Proc. Int. Conf. Learning Representations (2016) http://arxiv.org/abs/1604.03168.
[21] D. Miyashita, et al.: “Convolutional neural networks using logarithmic data representation” (2016). http://arxiv.org/abs/1603.01025.
[22] Z. Deng, et al.: “Reduced-precision memory value approximation for deep learning,” HP Laboratories, Tech. Rep. HPL-2015-100 (2015) https://www.labs.hpe.com/techreports/2015/HPL-2015-100.pdf.
[23] S. Han, et al.: “DSD: Regularizing deep neural networks with dense-sparse-dense training flow” (2016) http://arxiv.org/abs/1607.04381.
[24] L. Lai, et al.: “Deep convolutional neural network inference with floating-point weights and fixed-point activations” (2017) http://arxiv.org/abs/1703.03073.
[25] O. Russakovsky, et al.: “ImageNet large scale visual recognition challenge,” Int. J. Comput. Vis. 115 (2015) 211 (DOI: 10.1007/s11263-015-0816-y).
[26] Y. Jia, et al.: “Caffe: Convolutional architecture for fast feature embedding” (2014) http://arxiv.org/abs/1408.5093.
1 Introduction

Convolutional neural networks (CNNs) are being widely used in computer vision tasks [1, 2, 3, 4, 5, 6]. The achievement of the CNNs is based on their learning ability with their very deep structures. However, the deep structure requires an enormous amount of operations and storage, which make it hard to adopt CNNs to embedded devices. To overcome the limit of the embedded devices, there have been many studies to implement CNN accelerators in ASIC or FPGA [7, 8, 9, 10].

In ASIC or FPGA implementation, the most demanding part is the multiplication and accumulation (MAC) units and the internal SRAM. The amount of the resources depend much on the employed representation, which means that the number representation of the numerical values of the activations and the weights for implementation in this paper. The resource requirement can be reduced by using a short and simple representation for activations and weights in CNNs, but such representations degrade the accuracy. This leads to much effort to simplify the representation with acceptable degradation of the accuracy.

Previous studies, especially on ASIC or FPGA implementation, have usually exploited the 16-bit fixed-point representation because the fixed-point representation is believed to produce simpler operation circuits than the floating-point representation [7, 8, 9, 10]. Some other studies have focused on employment of different number representations for implementation of CNNs including shorter fixed-point representation [11, 12, 13, 14, 15, 16] and simpler representations like binary or ternary representations [17, 18]. The studies have reported no or a little accuracy loss, but they were tested with small data set [11, 12, 13, 14, 19] or with not-so-deep networks [15, 20, 21, 22, 23, 24]. The dynamic fixed scheme was sometimes used to make up for the insufficient dynamic range [14, 15, 16, 21]. When the accuracy of the neural network degraded severely, additional efforts were required such as fine-tuning or retraining [12, 13, 16].

The previous studies preferred the fixed-point representation because the implementation of the floating-point representation is complex and costly. That is true in CPU or GPU implementation, where only the predefined formats can be used. In ASIC or FPGA implementation, however, user-defined formats can be used including a shorter-width floating-point format.

In this letter, we propose a short floating-point representation and its adoption scheme to a CNN accelerator to achieve both low hardware cost and high accuracy without retraining or fine-tuning in the inference process of CNNs. With a few exponent bits, the width of the mantissa is reduced so that the total width is the half of the fixed-point representation. The reduced mantissa width and the small total width lead to smaller multipliers and memory, which also consume less energy. In spite of the reduced width, the accuracy degradation is under 1 percentage point on the CNN of 152 layers.

This letter is organized as follows. Section II and Section III explain the CNN and the number representation, and the short floating-point representation is proposed in Section IV. Section V shows the experimental results, and Section VI draws our conclusions.
2 Convolutional neural network and number representation

2.1 Convolutional neural network

A CNN usually consists of multiple convolutional layers and a few fully-connected layers. Between those layers, there are pooling and non-linear activation layers like rectified linear unit (ReLU) layers. Many recent CNNs exploit batch normalization layers to expedite the training process. Most of the computation in CNNs is performed in the convolutional layers, and the computation consists of the MAC operations between the input feature map and the weights. The entries of the input and output feature maps are called the input and output activations.

In ASIC implementation, the MAC operations of a convolutional layer are usually performed on a processing element (PE) consisting of multipliers and an adder tree [7]. Part of multiplications are calculated by multipliers, and the results are summed by the adder tree. Then the output of the adder tree is accumulated, and the accumulation result is transferred to the next layer. Some architectures have several PEs for parallel processing.

The CNN implementation also requires a large amount of memory for the weights and activations. In the usual ASIC or FPGA implementation, they are stored in an external DRAM, and internal SRAMs are used as caches.

2.2 Floating-point and fixed-point representation

Numerical values of activations and weights of a CNN can be represented using several formats, such as the fixed-point representation and the floating-point representation. The floating-point (Float) representation consists of a fixed number of significant digits called a mantissa, and a scale called an exponent, following the IEEE 754 standard. The 16-bit, 32-bit, and 64-bit representations, defined in the standard, are called half-precision, single-precision, and double-precision.

In the standard Float representation, the represented actual value, $x$, is $x = (-1)^{\text{sign}} \times \text{mantissa} \times 2^{\text{exponent-bias}}$. The standard also defines some exceptional cases: the denormalized number, infinities, and Not-a-Number (NaN). In the arithmetic circuits with the Float representation, the output usually uses the same format as the input.

In the fixed-point (Fixed) representation, a number is represented in the binary format with an implicit binary point. In this letter, the total width and the fractional part width will be written as WM and WF, respectively. The Fixed representation is simpler, but the representation usually has less dynamic range and precision than the Float representation. To alleviate the degradation by the deficiency, the output width of a multiplier or an adder tree is usually not reduced to the width of the input activation. The width is reduced when the output of the accumulation is transferred to the next layer. This scheme will be called the full precision internal operation in this letter.

In a CNN, it is known that using different WMs or WFs for several value groups enhances the accuracy. In [14], using different WFs is called the dynamic fixed scheme. In this letter, we use the dynamic fixed scheme only for the weights grouped layer-by-layer. Determining a WF for each group of values requires the analysis of the value distribution. The weights are known a priori once trained, so
there is a chance to find the near-optimal WFs from the distribution of the weight values. However, it is difficult to determine the distribution of the activation values \textit{a priori}. The analysis requires running the network with the training data, which may not be convenient in some environments. In an embedded system, for example, the training can be performed by a remote server continuously, and the trained weights are downloaded to the terminal embedded devices periodically. The activation distribution analysis should keep on being performed to follow the continuously changing weights, which requires much additional effort.

2.3 Representation width
Because of the simpler operation circuits, the Fixed representation, especially of 16 bits, has been usually adopted in the field of ASIC or FPGA accelerator implementations [7, 8, 9, 10]. The studies focusing on the representation format, on the other hand, have tried various representations including the Fixed representations, power-of-two representation, and shared values. The previous studies reported successful results with small widths, but they tested only small data sets like MNIST, CIFAR-10, or SVHN [11, 12, 13, 14, 16, 19] or required activation distribution analysis [16, 21]. Some studies exploited an uncommon base like \(\sqrt{2}\) [21]. Very short representations like the binary or ternary representation required network structure modification or severe retraining and could not be applied to the activations [17, 18]. An 8-bit floating-point representation has been tried in [22, 24], but it was applied only to the weights with the insufficient hardware cost analysis. The discussion of a short floating-point representation in [16] was done with a few bit width cases and no hardware cost analysis. The lack of case analyses resulted in severe degradation in the accuracy results.

Furthermore, the previous studies only tested networks with a depth of up to around 20, a few of AlexNet [1], NiN [2], GoogLeNet [3], VGG [4], ResNet-18 [5], and SqueezeNet [6]. The depth is much less than that of the state-of-the-art networks like ResNet [5] with depths of 50–152.

3 Short floating-point representation
In this section, we propose a short floating-point (S-Float) representation for CNNs. The S-Float representation is based on the standard Float representation, but it is simplified and width-reduced. The exponent concept is exploited for a large dynamic range, and the normalization with the implicit 1 scheme is used to save one more bit. The denormalized case is also used for very small values. The other complex cases like NaN and infinity are not used. The reduced width may cause the accuracy degradation, so the full precision internal operation is adopted as in the Fixed representation.

The S-Float representation is configured with three parameters, WM, WE and B. The WM is the bit width of the sign bit and the mantissa part, the WE is that of the exponent part, and the B is the exponent bias. The total width is WM+WE. In the standard Float representation, the binary point of the mantissa is located at the right of the implicit 1, but in the S-Float representation, it is located at the right of the whole mantissa to complement the reduced dynamic range.
3.1 Multiplication and accumulation

In the S-Float representation, the multiplication is performed on the implicit 1s and mantissas of the two operands as shown in Fig. 1. In the figure, “manA” and “manB” denotes the mantissas of the two operands. The multiplication result has a width of $W_{MA} + W_{MB}$. The sign is determined after the multiplication. The exponent parts, “expA” and “expB” in the figure, are added, and the result is used to shift the signed multiplication result. The multiplier output width is $W_{OM} = W_{MA} + W_{MB} + 2^{WE_A} + 2^{WE_B} - 1$.

In the standard Float representation, the result of an operation is usually converted back to the Float format. The conversion is not harmful in the standard Float representation because it has enough length of the mantissa and exponent. The S-Float representation, however, does not have long bit width, which can cause the conversion to degrade the performance.

To cope with the degradation, the full precision internal operation, usually used in the Fixed representation, is exploited. The output of a multiplier is transferred to the adder tree without width reduction or format conversion, and the internal operation of the adder tree is performed without width reduction, too. If the depth of the adder tree is $TD$, the output of the adder tree has a width of $W_{OM} + TD$, as shown in Fig. 2. The adder tree output is accumulated and then converted to the S-Float format.
3.2 Area of multipliers and memory

The memory size is directly related to the bit width. The Fixed representation
requires a large width to cover the dynamic range and precision. The S-Float
representation can achieve the range and precision with a smaller bit width because
a few bits in the exponent part can save a number of bits in the mantissa. For
example, the exponent part of a 3-bit width is almost equivalent to the mantissa
with 7 more bits.

The S-Float representation scheme can also reduce the area of the multiplier.
The circuit complexity of a multiplier is proportional to the product of the widths
of the operands. Since the multiplier’s operands are just the implicit 1s and
mantissas, the input bit width can be much smaller than that of the Fixed
representation. If we reduce the mantissa width by half, the multiplier complexity
is reduced to one-fourth.

The additional cost of adopting the S-Float representation is only the shifting
logic at the output of a multiplier and the formatting logic at the output of the
accumulation.

4 Experimental results

The experiment was performed in two ways: an accuracy simulation and a
hardware cost comparison. The simulation was done with state-of-the-art networks,
including a very deep one like ResNet-152 [5]. It will be shown that the S-Float
representation produces higher accuracy and lower complexity than the Fixed
representation. The experiments have been performed for CNNs, but the S-Float
representation is not limited to CNNs. CNNs are selected since they require
enormous amount of operations and storage, where the S-Float representation
may show more improvement.

4.1 Accuracy experiments

The top-5 accuracy analysis for the ImageNet 2012 validation data set [25] was
performed with various representations. The Caffe framework [26] was modified
to process the Fixed and S-Float representations. Numerical values of weights are
converted according to the target representation when the first batch is processed
and maintained in the following batches. Numerical values of activations are
converted at the input of the first layer, and processed in the target representation
at all of the layers in CNNs including convolutional layers, fully-connected layers,
ReLU layers, pooling layers, and batch-normalization layers. No retraining or fine-
tuning was performed after the quantization. The number of convolution and fully-
connected layers used in the employed networks are given at the second and third
rows of Table I. AlexNet is used with the local response normalization layers
removed.

Table I. CNNs for experiments

|      | AlexNet | VGG16 | SqueezeNet | ResNet-50 | ResNet-152 |
|------|---------|-------|------------|-----------|------------|
| Conv | 5       | 13    | 26         | 53        | 155        |
| FC   | 3       | 3     | 0          | 1         | 1          |
The acceptable degradation was assumed to be the 1 percentage point. First, the configuration of the activation representation was varied with sufficient width for weights. In the Fixed representation, WF was set to $\text{WM} = 4 + 0.5$, which was selected after experiments with $\text{WM} = 4 + 0.5$, $\text{WM} = 2 + 0.5$, $\text{WM} = 4 + 0.5$, and 0. In the S-Float representation, the parameter B, the bias in the exponent, was set to $2^{\text{WE}} - 1$ as in the standard Float representation.

Table II presents the top-5 accuracies for various recently-proposed networks. In the table, the SP Float row is the baseline of the single-precision Float representation. The S-Float representation is described with (WE,WM). The accuracy values within the 1-percentage-point criterion are marked with a boldface font. Whereas the Fixed representation requires more than 13 bits, the S-Float representation realizes an accuracy comparable to that of the SP Float with shorter widths. With $\text{WE} = 4$, $\text{WM} = 4$ or 5 obtains an accuracy better than or almost equal to that of SP-Float, where the total width is 8 or 9 bits including the sign bit. $\text{WE} = 5$ obtains a similar curve, which means 4 bits is enough for the exponent width. The VGG16 network requires more bits than the deeper networks probably because the VGG16 network has more diverse value distributions due to the lack of the batch normalization layers. In some layers of the VGG16 network, the maximum output value is around $2^{14}$, but in other layers, that value is around $2^7$. If the dynamic fixed scheme is used, VGG16 may present better results. Contrary to the long width required by the Fixed representation, we can notice that the S-Float representation requires just 8 or 9 bits in all the networks.

In addition to the achievement of the high accuracy with short width, the S-Float representation is little affected by the existence of the batch normalization layers or the value distribution. All the networks reach the target accuracy with the same 8–9 bit width. With the S-Float representation, we can apply the same representation configuration to various networks. This is an important property in the ASIC accelerator implementation. Once manufactured, an ASIC accelerator should be able to run various networks. With the S-Float representation, if an

| Representation | AlexNet | VGG16 | SqueezeNet | ResNet-50 | ResNet-152 |
|---------------|---------|-------|------------|-----------|------------|
| SP Float      | 79.964  | 88.366| 80.3899    | 90.5199   | 92.2014    |
| Fixed 18 bits | 79.9360 | 88.4142| 80.3919    | 91.1821   | 92.2622    |
| Fixed 16 bits | 79.9460 | 87.0542| 80.4159    | 91.0841   | 92.1982    |
| Fixed 15 bits | 79.9640 | 79.3121| 80.2319    | 91.1261   | 92.1722    |
| Fixed 14 bits | 79.8260 | 55.2919| 76.4380    | 90.9801   | 92.0181    |
| Fixed 13 bits | 79.8160 | 55.2740| 76.4240    | 90.8521   | 91.5962    |
| Fixed 12 bits | 77.5260 | 21.4640| 51.8640    | 89.0262   | 88.7743    |
| Fixed 11 bits | 65.0800 | 5.3580 | 14.2380    | 73.8400   | 72.0522    |
| S-Float (5,5) | 79.8200 | 88.3321| 79.8860    | 90.9822   | 92.0703    |
| S-Float (5,4) | 79.4340 | 88.0822| 78.4660    | 90.4461   | 91.3103    |
| S-Float (5,3) | 78.0960 | 86.9142| 72.4220    | 88.0643   | 85.0024    |
| S-Float (5,2) | 72.2660 | 82.3781| 50.5160    | 68.4539   | 25.8500    |
| S-Float (4,5) | 79.7680 | 88.3361| 79.9159    | 90.8741   | 91.9882    |
| S-Float (4,4) | 79.4780 | 88.1162| 78.5540    | 90.4722   | 91.2663    |
| S-Float (4,3) | 78.1040 | 84.5142| 72.3780    | 88.0122   | 84.7664    |
| S-Float (4,2) | 72.2760 | 57.0180| 48.4080    | 68.4900   | 25.4820    |
accelerator is designed with (WM, WE) = (4, 4) or (4, 5), it can run various networks without accuracy degradation.

Table III analyzes the accuracy with various weight representations. The activation used the S-Float representation with (WE, WM) = (4, 5). The dynamic fixed scheme was applied to the weight representation. If the largest absolute weight value in a layer is in range of $[2^{WE} \cdot 1 - x + WM, 2^{WE} - 1 + x]$ in the Fixed representation for the layer, and B is set to $2^{WE} - 1 - x + WM - 1$ in the S-Float representation. In the table, the 8-bit S-Float representation with (WE, WM) = (4, 4) experiences a slight degradation. The degradation is less than the 0.2 percentage point in VGG16 and ResNet-50 and around the 0.8 percentage point in SqueezeNet and ResNet-152. Actually, WE = 3 also has a similar accuracy for VGG16, SqueezeNet, and ResNet-50.

From the analysis in this section, the S-Float representation presents no or slight degradation with (WE, WM) = (4, 5) for activation and (WE, WM) = (4, 4) for weights. The activations are usually guaranteed to be positive because of the ReLU layers, so the actual bit width to be stored in a memory is 8 bits for both the activations and the weights. For the case that using the same representation is preferred, Table III also provides the accuracy with (WE, WM) = (4, 4) for both of them in the last row, where the accuracy degradation is less than 3 percentage points.

In AlexNet, the degradation with the S-Float representation is under 1 percentage points, which is much less than 4 percentage points, the degradation of [16]. In [16], only the case of (WE, WM) = (5, 3) was analyzed, showing such degradation. This letter, however, shows that better representations can be found by searching various widths. Furthermore, they did not analyze the hardware cost, another advantage of the S-Float representation.

### Table III.

| Representation | AlexNet | VGG16 | SqueezeNet | ResNet-50 | ResNet-152 |
|---------------|---------|-------|------------|-----------|------------|
| SP Float      | 79.964  | 88.3661 | 80.3899   | 90.5199   | 92.014     |
| Fixed 16 bits | 79.8160 | 88.3221 | 79.8939   | 90.9242   | 91.8682    |
| Fixed 14 bits | 79.8760 | 88.3081 | 79.8919   | 90.8341   | 91.3723    |
| Fixed 13 bits | 79.8620 | 88.3501 | 79.8519   | 90.9302   | 83.8324    |
| Fixed 12 bits | 79.8880 | 88.3201 | 79.9619   | 90.8841   | 5.1061     |
| Fixed 10 bits | 79.7780 | 88.3242 | 79.8759   | 90.3202   |            |
| Fixed 9 bits  | 79.8160 | 88.2481 | 79.7979   | 60.5199   |            |
| Fixed 8 bits  | 79.6240 | 88.1442 | 79.5879   | 10.5160   |            |
| Fixed 7 bits  | 79.1660 | 87.3701 | 78.9679   |            |            |
| Fixed 6 bits  | 75.7980 | 77.3020 | 76.4820   |            |            |
| S-Float (4,4) | 79.8300 | 88.2241 | 79.5999   | 90.4502   | 91.3789    |
| S-Float (4,3) | 79.5000 | 88.0001 | 79.0099   | 84.8782   | 86.9923    |
| S-Float (4,2) | 78.8980 | 87.2742 | 75.2040   | 57.6320   | 83.1703    |
| S-Float (3,4) | 79.7880 | 88.2681 | 79.6159   | 90.4142   | 0.5040     |
| S-Float (3,3) | 79.4980 | 87.9982 | 78.9860   | 85.0622   | 0.4900     |
| S-Float (3,2) | 78.9160 | 87.2022 | 75.1080   | 24.5520   | 0.4940     |
| S-Float (4,4)* | 79.4400 | 88.0302 | 78.2640   | 89.9602   | 89.9742    |

* With (WE, WM) = (4, 4) activation representation.
4.2 HW cost consideration

The Float representation is usually considered to have more complex operation circuits than the Fixed representation. To show that the S-Float representation can have simpler circuits, we implement a PE with 16 multipliers and a 16-input adder tree in the Fixed representation and the S-Float representation. In DianNao, a few memories and 16 PEs with the same configuration constitute a neural functional unit (NFU) [7]. The output activation is formatted with the same representation as the input activation. We synthesized the RTL implementations with Synopsys Design Compiler and the Synopsys 32/28 nm generic library. The synthesis script is described so that the circuit operates as fast as possible. Pipeline registers are inserted between the multipliers and the adder tree.

The synthesis results are compared in Table IV, where the first and second columns describe the representation of the activations and the weights. The S-Float representation is described with (WF, WM). The activations are assumed to be always positive, so the sign bit is ignored in the designs. Without the sign bit, the activations have the same bit width as the weights in all the rows of the table. The third to seventh columns compare the results of a PE: the area, the critical path delay, and the power consumption at 200 MHz.

| Activation Representation | Weight Representation | PE Area (µm²) | PE Delay (ns) |
|---------------------------|-----------------------|--------------|--------------|
| Fixed 17 bits             | Fixed 16 bits         | 52127        | 74654        | 5.26         |
| Fixed 13 bits             | Fixed 12 bits         | 30571        | 47153        | 4.54         |
| S-Float (4,6)             | S-Float (4,5)         | 20535        | 44783        | 5.44         |
| S-Float (4,5)             | S-Float (4,4)         | 17651        | 40622        | 4.56         |
| S-Float (4,4)             | S-Float (4,3)         | 14859        | 35866        | 4.68         |

| Activation Representation | Weight Representation | SRAM Area (µm²) | NFU Area (µm²) |
|---------------------------|-----------------------|---------------|---------------|
| Fixed 17 bits             | Fixed 16 bits         | 161734        | 100629        | 5573654      |
| Fixed 13 bits             | Fixed 12 bits         | 103942        | 67167         | 3436177      |
| S-Float (4,6)             | S-Float (4,5)         | 68586         | 46042         | 2392703      |
| S-Float (4,5)             | S-Float (4,4)         | 58321         | 39757         | 2041905      |
| S-Float (4,4)             | S-Float (4,3)         | 48818         | 33850         | 1707062      |

This is because the operands of the multipliers are only the mantissa part, which is shorter than the width of the Fixed representation. The S-Float representation produces slightly bigger adder trees because of the wider multiplier output after the shift by the exponent sum, but the area increase is not significant. In the comparison of the Fixed 17/16 bit and S-Float (4,5)/(4,4) for the activation/weight representation, the S-Float representation reduces the area of a PE by 34,032 µm² (45.6%). If 16 PEs are used as in DianNao, the reduction is multiplied by 16 to be 34,032 × 16 = 544,512 µm². The difference in the critical path delay is not so meaningful, and the power consumption is reduced with the S-Float representation, too.

The SRAM area is also compared in the table. The SRAMs are configured as in DianNao [7]. The NBin/SB/NBout buffers are the buffers for the input activations,
the weights, and the output activations, respectively. They are dual/dual/single-port SRAMs with 64 rows and 16/256/16 pieces of data per row, respectively. In DianNao, the widths of the activations and the weights are 16 bits. With the S-Float representation, the width of a piece of data can be reduced to 8 bits. The width reduction creates an SRAM area reduction by more than 50%. Furthermore, the width reduction may also reduce the DRAM access power by reducing the bit amounts to be transferred. With all the area reduction in logic and memories, the S-Float (4,5)/(4,4) representation is shown to have less NFU area by 63.4% than the Fixed 17/16-bit representation at the last column.

Fig. 3 shows the relation between the top-5 accuracy and the implementation area with various representations. A node with a pair of numbers indicates an S-Float representation with (WE,WM). The figure makes it clear that the S-Float representation is a more optimized one than the Fixed representation.

5 Conclusions

In this letter, we proposed the S-Float representation, a shortened and simplified Float representation, for CNN inference. Exploiting the exponent concept of the Float representation and the full precision internal operation of the Fixed representation, the proposed scheme simultaneously reaches high accuracy and low-complexity ASIC implementation. The 8-bit width S-Float representation shows no or slight accuracy degradation and more than a 60% area reduction. Furthermore, the proposed representation can be applied to the various networks with the same representation configuration. This is an important property in the ASIC implementation on which various networks should be performed once manufactured. For further research, the S-Float representation will be applied to the training process of CNNs.

Acknowledgments

This work was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2015R1D1A1A01058768). This work was also supported by IDEC (EDA Tool).