PAPER Special Section on Parallel, Distributed, and Reconfigurable Computing, and Networking

SENTEI: Filter-Wise Pruning with Distillation towards Efficient Sparse Convolutional Neural Network Accelerators

Masayuki SHIMODA[†a)], **Youki SADA**[†], **Ryosuke KURAMOCHI**[†], ***Nonmembers***, **Shimpei SATO**[†], ***and*** **Hiroki NAKAHARA**[†], ***Members***

SUMMARY In the realization of convolutional neural networks (CNNs) in resource-constrained embedded hardware, the memory footprint of weights is one of the primary problems. Pruning techniques are often used to reduce the number of weights. However, the distribution of nonzero weights is highly skewed, which makes it more difficult to utilize the underlying parallelism. To address this problem, we present SENTEI*, filter-wise pruning with distillation, to realize hardware-aware network architecture with comparable accuracy. The filter-wise pruning eliminates weights such that each filter has the same number of nonzero weights, and retraining with distillation retains the accuracy. Further, we develop a zero-weight skipping inter-layer pipelined accelerator on an FPGA. The equalization enables *inter-filter parallelism*, where a processing block for a layer executes filters concurrently with straightforward architecture. Our evaluation of semantic-segmentation tasks indicates that the resulting mIoU only decreased by 0.4 points. Additionally, the speedup and power efficiency of our FPGA implementation were 33.2× and 87.9× higher than those of the mobile GPU. Therefore, our technique realizes hardware-aware network with comparable accuracy.

key words: *sparse convolutional neural network, filter-wise pruning, distillation, FPGA*

1. Introduction

Convolutional neural networks (CNNs) [1] deliver state-of-the-art performance in computer-vision tasks such as object classification, object detection, and semantic segmentation. They have been increasingly required in a variety of embedded systems such as robots, self-driving cars, and drones. However, CNNs involve too many parameters (especially weights) for resource-constrained embedded hardware to store in its on-chip memory.

To address this problem, several network-compression techniques have been proposed. Pruning [2] is a compression technique that eliminates unnecessary weights from CNNs. Figure 1 details the procedure of pruning; it consists of three steps: training weights, pruning unnecessary weights, and retraining the remaining weights. Firstly, network connectivity (weights) are trained with a dataset. The resulting weight values indicate the magnitude of effect of each connection on the accuracy. Secondly, pre-trained weights below a threshold are pruned, and the unit is typically layer-by-layer. Finally, the remaining weights are retrained with the same dataset to achieve the original accuracy. The pruning technique leads to more than a 10-fold reduction in the amount of parameters with comparable accuracy [2]. However, as for the calculation latency, the utilization of high-degree parallelism becomes more difficult because the distribution of nonzero weights is skewed. The hardware requires fine-grained data parallelism with a flexible data path, which leads to a significant overhead.

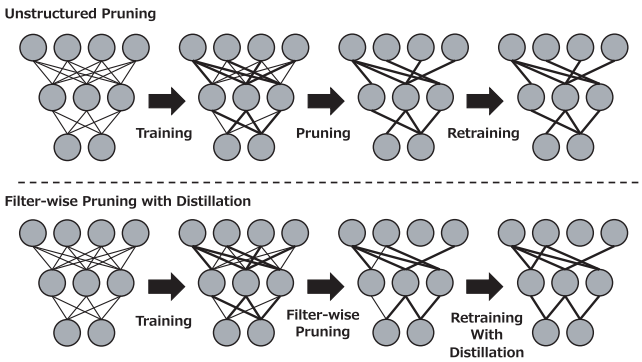


Fig. 1 Comparison with conventional unstructured pruning. Our filter-wise pruning makes neurons in a layer have the same number of connections (weights) from neurons in the previous layer.

To address this problem, we propose SENTEI, filter-wise pruning with distillation. At the pruning step, SENTEI prunes weights such that each filter has the same number of nonzero weights. Thus, each output neuron has the same number of connections with the previous ones, which makes it easier to utilize inter-filter parallelism. At the retraining step, the remaining weights are retrained with a distillation technique, where dense and sparse networks serve as the teacher and the student, respectively. By using the dense network's training information, the sparse network's accuracy can be increased and training convergence accelerated. Additionally, we develop a zero-weight skipping inter-layer pipelined accelerator on an FPGA. Because each filter has the same number of nonzero weights, utilizing inter-filter parallelism does not require complicated functions. Therefore, our architecture achieves a high degree of parallelism with simple hardware complexity.

The contributions of this study are summarized below.

Manuscript received January 7, 2020.
Manuscript revised May 21, 2020.
Manuscript publicized August 3, 2020.

[†]The authors are with Tokyo Institute of Technology, Tokyo, 152–8550 Japan.
a) E-mail: shimoda@reconf.ict.e.titech.ac.jp

DOI: 10.1587/transinf.2020PAP0013

*SENTEI is a Japanese word that means pruning. SENTEI not only trims a tree to make it look better, but also allows it to utilize nutrients efficiently. We decided this name since the effect is very similar to our technique's effect to CNN.

1. We propose SENTEI, filter-wise pruning with distillation. With comparable accuracy, it performs pruning such that each filter has the same number of weights to achieve a high degree of parallelism.
2. We develop a zero-weight skipping inter-layer pipelined accelerator that utilizes inter-filter parallelism. The accelerator realizes a high degree of parallelism while maintaining hardware simplicity.
3. We evaluate SENTEI based on a semantic-segmentation task. The results indicate that the accuracy is comparable with those of its dense-network counterparts.
4. We compare our system on an FPGA with a mobile GPU. The latency speedup and energy efficiency of the FPGA were 33.2× and 87.9× better than those of the mobile GPU.

This paper is an extension of a previous publication [3].

The rest of the paper is organized as follows: Sect. 2 provides motivation of our work. Section 3 explains a basic knowledge used throughout this paper. Section 4 introduces our proposed pruning technique, SENTEI (filter-wise pruning with distillation). Section 5 explains our accelerator for filter-wise pruned CNN. Section 6, 7, and 8 are experimental results, related works, and conclusion, respectively.

2. Motivation

Pruning is often used to reduce the number of weights for CNN implementation in resource-constrained embedded hardware. The procedure of the typical pruning technique [2] is given below:

1) Train the network weights
2) Prune the small weights below a threshold
3) Retrain the remaining weights

After pruning, most weights become zero; the weights are stored in a format such as the coordinate (COO) format or the compressed sparse row (CSR) format. As an example, we consider the pseudo-code shown in Fig. 2 for sparse convolution (CONV) with the COO format. The code consists of four loops. In each loop, a weight and an address (row, column, and channel values) are loaded, and the multiply-accumulate (MAC) operation is performed as many times as the number of non-zero weights. We categorize parallelism into the following different types based on the unrolling strategies.

```
L1:for(ofm_ch=0; ofm_ch < OFM_CH; ofm_ch++)
L2:for(ofm_h=0; ofm_h < OFM_H; ofm_h++)
L3:for(ofm_w=0; ofm_w < OFM_W; ofm_w++)
L4:for(n=0; n < NONZERO_W[ofm_ch]; n++)
  k_r   = COO_K_ROW[ofm_ch][n]
  k_c   = COO_K_COL[ofm_ch][n]
  k_ch  = COO_K_CH[ofm_ch][n]
  w = COO_W[ofm_ch][n]
  x = IFM[k_ch][ofm_h + k_c][ofm_w + k_r]
  OFM[ofm_ch][ofm_h][ofm_w] += x * w
```

Fig. 2 Pseudo-code of sparse CONV with the COO format.

1. Spatial Parallelism, where loop L2, loop L3, or both are unrolled
2. Inter-filter Parallelism, where loop L1 is unrolled
3. Intra-filter Parallelism, where loop L4 is unrolled

To realize high-speed calculation, we must utilize the above three types of parallelisms flexibly based on network architectures.

Figure 3 shows the nonzero-weight ratio of each pruned filter in a customized AlexNet [4]. There is a significant difference in the nonzero-weight ratios between a certain pair of filters. Similar to the case in the example, certain pruned networks have highly skewed nonzero-weight distributions [5]. Owing to the imbalance, it becomes more difficult to utilize inter- and intra-filter parallelisms. To address this problem, some studies have developed exclusive compilers or accelerators. However, they incur high development costs or feature complicated architecture. Our key idea is to equalize the number of nonzero weights of each filter during the training phase. This equalization makes it easier to exploit the underlying inter- and intra-filter parallelisms without complicated techniques.

3. Preliminary

3.1 Convolution (CONV)

Let $X^i(w,h,ch)$ be an input feature map (IFM) value at (w,h,ch) coordinates in i-th layer, $W^i(r,c,ch)$ be a weight value, w_{bias} be a bias, $S^i(w,h)$ be an intermediate variable, $f_{act}(x)$ be an activation function, and $X^{i+1}(w,h)$ be an $(i+1)$-th IFM, or output feature map (OFM). The $KW_i \times KH_i$ 2D CONV at (w,h) coordinates in $(i+1)$-th layer is as follows:

$$S^{i+1}(w,h) = w_{bias} + \sum_{ch=1}^{CH_i}\sum_{c=1}^{KH_i}\sum_{r=1}^{KW_i} X^i(w+r,h+c,ch) \times W^{i+1}(r,c,ch)$$
$$X^{i+1}(w,h) = f_{act}(S^{i+1}(w,h)),$$

where CH_i is the i-th IFM channels, r, c, and ch are coordinates of width-axis, height-axis, and channel-axis, respectively. The input is multiplied by a weight and the resulting sum is applied to the activation function. In this paper, we use the rectified linear unit (ReLU) functions as the activation function. When $KH_i = KW_i = 1$ (or pixel-wise CONV), it plays the role of classification corresponding to the part of input images.

3.2 Sparse CONV

After a pruning technique is applied, many weight values become zero. To store such weight parameters formed in the sparse matrix to block RAMs (BRAMs) efficiently, we employ a coordinate (COO) format. The arrays, *row*, *col*, *ch*, and *weight* store its row, column, channel indices, and nonzero-weights of the sparse matrix, respectively. Since

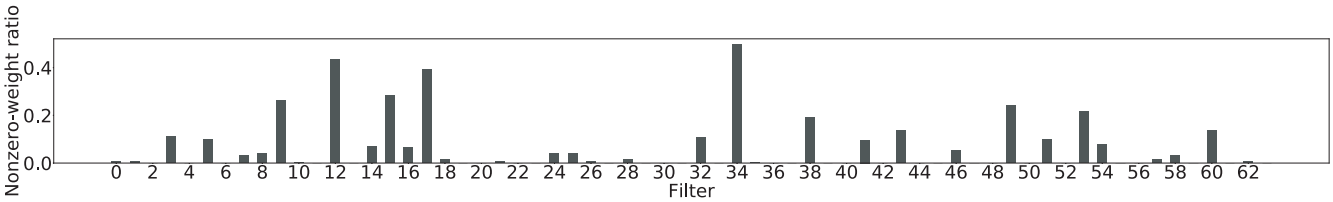


Fig. 3 Non-zero weight ratio of pruned filters in the first layer in our AlexNet. The layer has 64 kernels in 11×11 size, and sparse ratio is set 94%.

pruning realizes a high-sparse matrix, the overhead of additional memory requirement (*row*, *col*, and *ch* arrays) is not critical. When COO format is introduced, the sparse CONV at (w, h) coordinates is formulated as follows.

$$S^{i+1}(w,h) = w_{bias} + \sum_{j=1}^{N} X^i(w + r_j, h + c_j, ch_j) * w_j$$
$$X^{i+1}(w,h) = f_{act}(S^{i+1}(w,h)),$$

where N is the number of nonzero-weights, w_j is a weight value, and col_j and row_j are column and row indices, respectively.

4. SENTEI

In this paper, we present SENTEI, a filter-wise pruning with distillation. SENTEI consists of two steps: filter-wise pruning and retraining with distillation. The procedure is detailed below:

1) Train network weights
2) Sort the trained weights of **each filter** by their absolute values
3) Prune the small weights by a preset percent **filter-by-filter**
4) Retrain the remaining weights **with distillation**

By following the above procedure, each filter is the same sparse ratio, which simplifies the pruned-CNN accelerator. Moreover, retraining the weights with distillation retains the accuracy. We note that the retraining time does not change from conventional pruning [2]. The distillation scheme is detailed in the next section.

4.1 Distillation Scheme for Retraining Weights

Distillation [6] is a compression technique, where a network (student network) is trained using the outputs of a larger network (teacher network) to imitate the same outputs. The outputs of a teacher networks are used to improve the accuracy of a smaller student network. This study exploits this concept to obtain a highly sparse network while maintaining the accuracy. **To the best of our knowledge, this is the first study to implement the distillation technique for retraining weights.** It accelerates the convergence while increasing the accuracy of the sparse network.

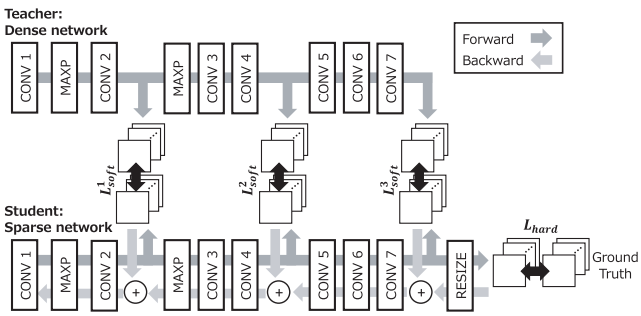


Fig. 4 Distillation scheme.

Figure 4 presents an outline of the distillation scheme with the AlexNet-based network that we have used. We assign a dense network as the teacher and a sparse network as the student. A training objective function L_{train} is written as:

$$L_{train} = \frac{1}{M} \sum_{m=1}^{M} L^m_{soft} + \beta L_{hard},$$

where M is the number of connections between the teacher and the student networks (this paper sets M 3), β is a hyper-parameter to balance their losses. L^m_{soft} is the m-th mean squared loss using both the network's feature map (FM), and L_{hard} is the pixel-wise cross entropy loss using ground truth. L^m_{soft} is defined by

$$L^m_{soft} = \frac{\alpha_m}{CH_i H_i W_i} \sum_{ch=1}^{CH_i} \sum_{h=1}^{H_i} \sum_{w=1}^{W_i} \left(X^i_s(w,h,ch) - X^i_t(w,h,ch) \right)^2,$$

where α_m is a hyper parameter to balance their losses, W_i, H_i, and CH_i indicate the i-th FM's width, height, and channel. $X^i_s(w,h,ch)$ and $X^i_t(w,h,ch)$ are the i-th FM values at (w,h,ch) in the student and the teacher networks, respectively. L_{hard} changes by a task, and in this paper we employ a semantic segmentation task for evaluation. Therefore, L_{hard} is pixel-wise cross entropy loss and defined by

$$L_{hard} = -\frac{1}{H_{in} W_{in}} \sum_{ch=1}^{\#class} \sum_{h=1}^{H_{in}} \sum_{w=1}^{W_{in}} \log\left(\sigma(w,h,ch)\right) p(w,h,ch),$$

where H_{in} and W_{in} denote input image size, $\sigma(w,h,ch)$

is predicted class probabilities, and $p(w,h,ch) \in \{0,1\}$ is the ground truth class probabilities at (w,h,ch) coordinates.

5. Hardware Implementation

Figure 5 shows the overall architecture. The design consists of buffer parts for parameters and the convolutional block (CB) parts, which constitute the inter-layer pipelined architecture. A combination of SENTEI and inter-layer pipelining enables a high degree of parallelism, together with architectural simplicity, because each layer has only one filter loop count owing to SENTEI. All parameters, including weights and biases, are loaded into on-chip buffers from DDR3 memory. Thereafter, the processor sends an input image to the design, and in the first CB, CONV operations are performed. The output feature maps are sent to the ping-pong buffer in the next CB to be read to compute the next layer operation. Finally, the outputs of the last CB (7-th CB in this study) are sent to the ARM processor. Weight matrices are stored in on-chip memory in the COO format.

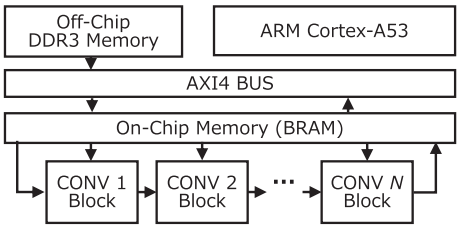


Fig. 5 Overall architecture.

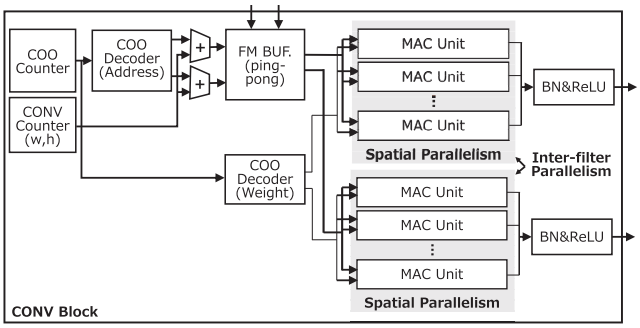


Fig. 6 Convolutional block architecture.

5.1 Convolutional Block

Figure 6 shows the architecture of a convolutional block (CB). It consists of a COO decoder, a COO counter, a counter for convolutional operation, a ping-pong buffer for feature maps (FMs), MAC units, and a processing element (PE) for both BN and ReLU. The calculations of MAC and PE are in the half-precision floating-point representation.

The COO counter counts the number of non-zero weights of each filter; the COO decoder obtains the number and outputs the corresponding relative address or row, column, and channel values. Next, to fetch the FM values, the absolute address is calculated using both the relative address and the convolutional counter, which indicates the coordinates where the CONV operation is performed. Figure 7 presents an example of the functionality. The CONV coordinates are $(w,h) = (1,1)$, and the relative address is $(row,col,ch) = (1,2,0)$. Therefore, the absolute coordinates of the corresponding FM value are $(w+row, h+col, ch) = (2,3,0)$. The fetched value and the weight are fed into a multiply-accumulate (MAC) unit, followed by PE for BN and ReLU. Finally, the output is stored in the next ping-pong buffer in the next CB.

This study employs spatial and inter-filter parallelisms. The parallelism improves throughput by at least 2× while the utilization of BRAMs little increases. Since exploiting inter-filter parallelism processes with two filters concurrently, the expected improvement of throughput is 2× at least, which implies that high-level synthesis may optimize the inter-filter parallel architecture automatically. As for BRAMs, there are two types of BRAMs in our architecture: for FMs and COO formatted arrays. Using inter-filter parallelism does not change the number of BRAMs for feature maps since we only change the BRAM configuration from single-port mode to double-port mode. On the other hand,

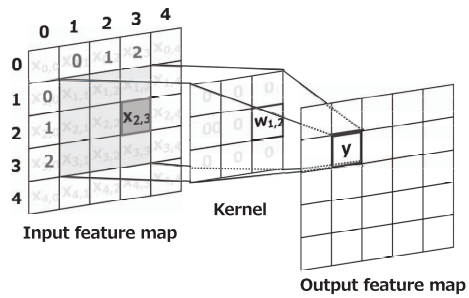


Fig. 7 Absolute and relative coordinates. The CONV counter has the absolute coordinates of the top-left pixel in the kernel, such as $(w,h) = (1,1)$ and the COO decoder contains relative coordinates $(row,col,ch) = (1,2,0)$.

since the previous architecture uses distributed single-port LUTs to store COO formatted arrays, we change these LUTs to dual-pot BRAMs, leading to an increase of the BRAM utilization. Thus, we expect throughput is at least 2×, and the BRAM utilization increases. From two coordinates of nonzero weights in a certain pair of filters, several successive FM values along to the width-axis are fetched and sent simultaneously. The corresponding weights are broadcast to the corresponding MAC units. As for COO indices, we used true dual port BRAMs as COO index buffers to read two corresponding indices in one cycle.

6. Experimental Results

In our experiments, we used an AlexNet-based network for the following reasons: 1) pruned AlexNet has a highly skewed distribution of nonzero weights, which means that filter-wise pruning without distillation leads to significant accuracy degradation. By using AlexNet, we can evaluate

the effectiveness of SENTEI in preventing it. 2) AlexNet is very useful for efficient edge-device implementation because big kernels in the first two layers reduce the input FM resolution; buffer size of the FM is one of the primary problems in the implementation of resource-constrained hardware. The network architecture is presented in Table 1. In this study, the total sparse ratio was 93.6%, and sparse ratio of the last layers was lower than that of the first layers to maintain accuracy.

6.1 Semantic Segmentation

Semantic segmentation is a fundamental task in image processing that performs pixel-wise classification, as shown in Fig. 8. In this paper, we employ semantic segmentation tasks to evaluate our proposal. We employ three metrics that are often used to measure performance. Let n_{ij} is the number of pixels of class i predicted as class j. The metrics are as follows:

1. Pixel-wise accuracy (PwiseAcc), $\frac{\sum_{i=1}^{\#class} n_{ii}}{\sum_{i=1}^{\#class}\sum_{j=1}^{\#class} n_{ij}}$.
2. Mean class accuracy (mClassAcc), $\frac{1}{\#class}\sum_{i=1}^{\#class}\frac{n_{ii}}{\sum_{j=1}^{\#class} n_{ij}}$.
3. Mean intersection over union (mIoU), $\frac{1}{\#class}\sum_{i=1}^{\#class}\frac{n_{ii}}{\sum_{j=1}^{\#class}(n_{ij}+n_{ji})-n_{ii}}$.

Fig. 8 Semantic segmentation.

Table 1 Sparse AlexNet-based network architecture.

Layer type	IFM CH	OFM CH	INF Size	Kernel Size	Stride	Padding	Zero-weight Ratio[%]
Hardware part:							
CONV	3	64	480×360	11×11	4	0	94.2
MAXP	64	64	119×89	3×3	2	0	-
CONV	64	64	59×44	5×5	1	2	94.1
MAXP	64	64	59×44	3×3	2	0	-
CONV	64	128	29×22	3×3	1	1	94.1
CONV	128	128	29×22	3×3	1	1	94.0
CONV	128	128	29×22	3×3	1	1	94.0
CONV	128	128	29×22	1×1	1	0	80.5
CONV	128	11	29×22	1×1	1	0	75.8
Software part:							
Resize	11	11	22×29	-	-	-	-
Total	-	-	-	-	-	-	93.6

6.2 Pruning Technique Evaluation

We used the CamVid [7] dataset, which is one of the popular datasets for the semantic-segmentation task. The image size is set as $(width, height) = (480, 360)$, and the number of the categories is 11. Our network cannot be compared with other networks because there is considerable difference in their network architectures. Therefore, we used the dense AlexNet-based network in floating precision as the baseline network.

Table 2 presents the results of an accuracy comparison. Our sparse model delivered performance comparable to that of a dense model. The results indicate that SENTEI realizes a high-sparse network with negligible accuracy degradation.

6.3 Comparison of FPGA Implementation with Various Image Sizes

We implemented our network for various image sizes on an FPGA by using Xilinx Inc. SDSoC 2017.4 with a timing constraint of 99.9 MHz. The FPGA board is a Xilinx Inc. Zynq UltraScale+ MPSoC zcu102 evaluation one, equipped with a Xilinx Zynq UltraScale+ MPSoC FPGA (ZU9EG, 68,520 Slices, 269,200 FFs, 1,824 18Kb BRAMs, 2,520 DSP48Es). Table 3 presents the implementation results. To increase the memory bandwidth, our architecture uses array partitioning and ping-pong buffers for FMs; therefore, the utilization of the BRAM is dominant, and our architecture cannot realize the implementation for an image size larger than $(300, 225)$. Additionally, inter-filter parallelism replaces distributed LUTs with BRAMs, increasing the BRAM utilization from the previous implementation.

6.4 Platform Comparison

We compared the performances of the FPGA and a Jetson

Table 3 Resource utilization on the zcu102 evaluation board using SDSoC 2017.4.

	120 × 90	180 × 135	240 × 180	300 × 225
18Kb BRAM	13.38	25.88	44.96	72.26
DSP48E	12.10	18.13	24.68	30.36
FF	10.07	13.87	17.41	21.32
LUT	22.94	30.43	37.03	44.65

Table 2 Semantic segmentation result.

	SpRatio	PwiseAcc	mClassAcc	mIoU	Sky	Build.	Pole	Road	Pavement	Tree	Sign	Fence	Car	Ped.	Bicyclist
Dense	0.0	78.2	66.3	45.3	85.8	55.5	9.0	86.7	63.8	58.5	12.8	17.3	63.3	21.0	24.3
Layerwise [2]	93.6	72.1	64.4	40.1	83.8	39.5	7.0	84.3	62.5	55.9	10.0	13.8	57.2	14.2	14.1
Filterwise	93.6	70.6	62.2	38.5	83.2	38.1	7.0	82.3	58.0	55.6	8.3	13.9	51.7	11.7	13.9
Filterwise+distil.	93.6	79.4	61.6	**44.9**	85.8	60.8	8.3	84.2	59.3	60.7	16.4	13.6	61.5	19.0	24.5

Table 4 Comparison with a mobile GPU with a image in 300×225 size

	Mobile GPU	FPGA
Platform	Jetson TX2	zcu102
Clock Freq. [GHz]	1.3	0.1
Speed [avg. FPS]	16.3	**541.5**
Power [W]	2.9	1.1
Efficiency [FPS/W]	5.6	**492.3**

TX2 GPU (NVIDIA Corp.). Note that we used the same sparseness of CNN for both the platforms for a fair comparison, and the Chainer framework [8] was used for the GPU implementation. Table 4 presents the performance-comparison results. Power consumption is dynamic power consumption. The inter-filter parallelism and auto optimization of high-level synthesis improved throughput more than 2× from the previous implementation. In comparison with the mobile GPU, FPGA reduced the power by 1.1 watts, and increased the speed 33.2×, which meets the real-time processing requirement (30 FPS). In terms of performance per power consumption (FPS/W), the FPGA was 87.9 times better than its mobile GPU counterpart.

7. Related Works

7.1 Sparseness Approach for Weight Memory Reduction

A modern CNN requires a large number of weight parameters, and an FPGA on-chip memory cannot store all the parameters. To address this issue, there are two types of sparseness (pruning) methods for after and during training.

As for pruning during training, [9] proposed a gradual pruning method that increases the weight pruning ratio using a binary mask variable. In [10], various sparse dropout strategies were used for both fully connected and convolutional layers to realize high sparsity, and in [11], the rank of the parameter matrix in each layer was reduced.

In contrast, there are many studies conducted on pruning after training, and the representative one is deep compression [2], which combines quantization, weight pruning, and Huffman coding method to achieve a 3-4× speedup. For the hardware of SIMD architecture, [12] proposed a SIMD-aware weight pruning that provides high parallelism using a specific format such as the compressed sparse rows (CSR) format.

We propose SENTEI, which eliminates weights filter-by-filter by a preset percent after training. From our preliminary experiments, it makes training convergence faster than those of pruning while training.

7.2 Sparse Convolutional Network Architecture

Our sparsity approach has attracted increasing interests as it reduces the data footprint and eliminates computation, and it achieves a good balance between compression ratio and accuracy. The following are three types of architectures based on the values that are skipped: weights, activations, and both.

7.2.1 Zero-Weight Skipping Architecture

It eliminates the zero-weight calculations only. Cambricon-X [13] allows weight buffers in different processing elements to load new data from the memory asynchronously to improve the overall efficiency since the numbers of weights of different neurons may differ significantly. J. Li, et al. [5] proposed a concise convolution rule (CCR) that decomposes sparse kernels into multiple effective non-zero sub-kernels using triplet representation.

7.2.2 Zero-Activation Skipping Architecture

It skips zero-activation calculations only. CNVLUTIN [14] architecture skips zero-activation multiplications to improve both performance and energy efficiency.

7.2.3 Zero-Weight and -Activation Skipping Architecture

It skips all inefficient computations. The EIE [15] exploits both sparsity and the indexing overhead. PermDNN [16] outperforms the EIE by generating a hardware-friendly structured sparse neural network using permuted diagonal matrices. Cambricon-S [17] introduces block pruning to reduce the irregularity. The SCNN [18] achieves a performance improvement of 2.7× with Cartesian product-based processing. STICKER [19] applied two-way set associative PEs to SCNN architecture, thereby reducing the memory area to 92%. Wang, et al. [20] improved the hash-based accelerator for SCNN using a load-balancing algorithm, and SpAW [21] proposed a dual-indexing module to utilize the sparsity more efficiently. For the efficient utilization of the SIMD architecture, B. Lai et al. [22] proposed weight rearrangement that parses through the weights at compile time and redistributes the nonzero weights to balance the computation.

We developed a zero-weight skipping architecture for filter-wise pruned CNNs because the zero-weight skipping architecture has achieved the best trade-off between speedup and hardware-complexity as mentioned in [5]. The primary difference from conventional architectures is not multiplexer-based architecture, but BRAM-based architecture. While the accelerator proposed by [13] is a multiplexer-based architecture, our accelerator is a BRAM-based architecture for random access to corresponding IFMs, which is an important contribution to efficient FPGA implementation from the view point of area efficiency. Thus, we decided that a pruning unit is a filter-wise unit to use BRAMs efficiency, and our architecture is more optimal for FPGA implementation of sparse CNNs.

8. Conclusion

We presented SENTEI, filter-wise pruning with distillation, to realize hardware-aware network architecture while retaining a comparable accuracy. Equalizing the number of

non-zero weights in each filter allowed the easy utilization of inter- and intra-filter parallelisms, and retraining weights with distillation retained the accuracy. We believe that our technique achieves a better trade-off between accuracy and network complexity in comparison with conventional pruning techniques.

Acknowledgments

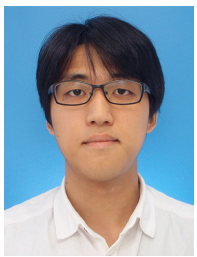
This research is supported in part by the Grants in Aid for Scientistic Research of JSPS, Industry-academia collaborative R&D programs centre of innovation (COI) program, Core Research for Evolutional Science and Technology (CREST), and the New Energy and Industrial Technology Development Organisation (NEDO). Also, thanks to the Xilinx University Program (XUP), Intel University Program, and the NVIDIA Corp.'s support.

References

[1] Y. Lecun, Y. Bengio, and G. Hinton, "Deep learning," Nature, vol.521, pp.436–444, 5 2015.
[2] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," Advances in neural information processing systems, pp.1135–1143, 2015.
[3] M. Shimoda, Y. Sada, and H. Nakahara, "Filter-wise pruning approach to FPGA implementation of fully convolutional network for semantic segmentation," Proc. 15th International Symposium on Applied Reconfigurable Computing , pp.371–386, Darmstadt, Germany, April 2019.
[4] A. Krizhevsky, I. Sutskever, and G.E. Hinton, "Imagenet classification with deep convolutional neural networks," Proc. 25th Int. Conf. Neural Information Processing Systems - Volume 1, NIPS'12, (USA), pp.1097–1105, Curran Associates Inc., 2012.
[5] J. Li, G. Yan, W. Lu, S. Jiang, S. Gong, J. Wu, and X. Li, "Ccr: A concise convolution rule for sparse neural network accelerators," 2018 Design, Automation Test in Europe Conference Exhibition (DATE), pp.189–194, March 2018.
[6] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," arXiv preprint arXiv:1503.02531, 2015.
[7] G.J. Brostow, J. Fauqueur, and R. Cipolla, "Semantic object classes in video: A high-definition ground truth database," vol.30, no.2, pp.88–97, Jan. 2009.
[8] S. Tokui, K. Oono, S. Hido, and J. Clayton, "Chainer: a next-generation open source framework for deep learning," Proc. Workshop on Machine Learning Systems (LearningSys) in The Twenty-ninth Annual Conference on Neural Information Processing Systems (NIPS), pp.1–6, 2015.
[9] M. Zhu and S. Gupta, "To prune, or not to prune: exploring the efficacy of pruning for model compression," CoRR, vol.abs/1710.01878, 2017.
[10] D. Molchanov, A. Ashukha, and D. Vetrov, "Variational dropout sparsifies deep neural networks," arXiv preprint arXiv:1701.05369, 2017.
[11] J.M. Alvarez and M. Salzmann, "Compression-aware training of deep networks," Advances in Neural Information Processing Systems, pp.856–867, 2017.
[12] J. Yu, A. Lukefahr, D. Palframan, G. Dasika, R. Das, and S. Mahlke, "Scalpel: Customizing dnn pruning to the underlying hardware parallelism," Proc. 44th Annual International Symposium on Computer Architecture, ISCA '17, (New York, NY, USA), pp.548–560, ACM, 2017.
[13] S. Zhang, Z. Du, L. Zhang, H. Lan, S. Liu, L. Li, Q. Guo, T. Chen, and Y. Chen, "Cambricon-x: An accelerator for sparse neural networks," 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), pp.1–12, Oct. 2016.
[14] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N.E. Jerger, and A. Moshovos, "Cnvlutin: Ineffectual-neuron-free deep neural network computing," 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA), pp.1–13, June 2016.
[15] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M.A. Horowitz, and W.J. Dally, "Eie: efficient inference engine on compressed deep neural network," 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA), pp.243–254, IEEE, 2016.
[16] C. Deng, S. Liao, Y. Xie, K.K. Parhi, X. Qian, and B. Yuan, "Permdnn: Efficient compressed dnn architecture with permuted diagonal matrices," 2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), pp.189–202, Oct. 2018.
[17] X. Zhou, Z. Du, Q. Guo, S. Liu, C. Liu, C. Wang, X. Zhou, L. Li, T. Chen, and Y. Chen, "Cambricon-s: Addressing irregularity in sparse neural networks through a cooperative software/hardware approach," 2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), pp.15–28, Oct. 2018.
[18] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S.W. Keckler, and W.J. Dally, "Scnn: An accelerator for compressed-sparse convolutional neural networks," 2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA), pp.27–40, June 2017.
[19] Z. Yuan, J. Yue, H. Yang, Z. Wang, J. Li, Y. Yang, Q. Guo, X. Li, M. Chang, H. Yang, and Y. Liu, "Sticker: A 0.41-62.1 tops/w 8bit neural network processor with multi-sparsity compatible convolution arrays and online tuning acceleration for fully connected layers," 2018 IEEE Symposium on VLSI Circuits, pp.33–34, June 2018.
[20] J. Wang, Z. Yuan, R. Liu, H. Yang, and Y. Liu, "An n-way group association architecture and sparse data group association load balancing algorithm for sparse cnn accelerators," Proc. 24th Asia and South Pacific Design Automation Conference, ASPDAC '19, (New York, NY, USA), pp.329–334, ACM, 2019.
[21] C.-Y. Lin and B.-C. Lai, "Supporting compressed-sparse activations and weights on simd-like accelerator for sparse convolutional neural networks," Proc. 23rd Asia and South Pacific Design Automation Conference, ASPDAC '18, (Piscataway, NJ, USA), pp.105–110, IEEE Press, 2018.
[22] B. Lai, J. Pan, and C. Lin, "Enhancing utilization of simd-like accelerator for sparse convolutional neural networks," IEEE Trans. Very Large Scale Integration (VLSI) Systems, vol.27, no.5, pp.1218–1222, May 2019.

Masayuki Shimoda received the B.E. and M.E. degree in engineering from Tokyo Institute of Technology, Tokyo, Japan, in 2018 and 2020, respectively. In 2020, he joined NTT Network Innovation Laboratories, Nippon Telegraph and Telephone Corporation, Yokosuka, Japan. His current research interests include Deep Neural Network and FPGA.

Youki Sada received the B.E. degree from National Institute of Technology Tokyo College in 2018, and the M.E. degree from Tokyo Institute of Technology in 2020. He is currently working at NEC Corp. and his research interests include an image processing by Deep Neural Network and FPGA.



Ryosuke Kuramochi received the B.E. degree in engineering from Tokyo Institute of Technology, Tokyo, Japan, in 2020. He is currently a Master Student with the Department of Information and Communications Engineering of Tokyo Institute of Technology. His current research interests include Deep Neural Network and FPGA.

Shimpei Sato received the B.E., M.E., and D.E. degrees in engineering from Tokyo Institute of Technology, Tokyo, Japan, in 2007, 2009, and 2014, respectively. He is currently an Assistant Professor with the Department of Information and Communications Engineering of Tokyo Institute of Technology. From 2014 to 2016, he worked in High performance computing area as a post doctoral researcher, where he investigated an application performance analysis/tuning method. His current research interests include approximate computing realization by architecture design and circuit design and their applications. He is a member of IEEE and IPSJ.

Hiroki Nakahara received the B.E., M.E., and Ph.D. degrees in computer science from Kyushu Institute of Technology, Fukuoka, Japan, in 2003, 2005, and 2007, respectively. He has held research/faculty positions at Kyushu Institute of Technology, Iizuka, Japan, Kagoshima University, Kagoshima, Japan, and Ehime University, Ehime, Japan. Now, he is an associate professor at Tokyo Institute of Technology, Japan. He was the Workshop Chairman for the International Workshop on Post-Binary ULSI Systems (ULSIWS) in 2014, 2015, 2016 and 2017, respectively. He searved the Program Chairman for the International Symposium on 8th Highly-Efficient Accelerators and Reconfigurable Technologies (HEART) in 2017. He received the 8th IEEE/ACM MEMOCODE Design Contest 1st Place Award in 2010, the SASIMI Outstanding Paper Award in 2010, IPSJ Yamashita SIG Research Award in 2011, the 11st FIT Funai Best Paper Award in 2012, the 7th IEEE MCSoC-13 Best Paper Award in 2013, and the ISMVL2013 Kenneth C. Smith Early Career Award in 2014, respectively. His research interests include logic synthesis, reconfigurable architecture, digital signal processing, embedded systems, and machine learning. He is a member of the IEEE, the ACM, and the IEICE.