Flexible Group-Level Pruning of Deep Neural Networks for On-Device Machine Learning

Kwangbae Lee, Hoseung Kim, Hayun Lee, Dongkun Shin
Department of Electrical and Computer Engineering, Sungkyunkwan University, Korea
{kblee93, ghtmd123, lhy920806, dongkun}@skku.edu

***Abstract*—Network pruning is a promising compression technique to reduce computation and memory access cost of deep neural networks. Pruning techniques are classified into two types: fine-grained pruning and coarse-grained pruning. Fine-grained pruning eliminates individual connections if they are insignificant and thus usually generates irregular networks. Therefore, it is hard to reduce model execution time. Coarse-grained pruning such as filter-level and channel-level techniques can make hardware-friendly networks. However, it can suffer from low accuracy. In this paper, we focus on the group-level pruning method to accelerate deep neural networks on mobile GPUs, where several adjacent weights are pruned in a group to mitigate the irregularity of pruned networks while providing high accuracy. Although several group-level pruning techniques have been proposed, the previous techniques select weight groups to be pruned at group-size-aligned locations. In this paper, we propose a more flexible approach, called unaligned group-level pruning, to improve the accuracy of the compressed model. We can find the optimal solution of the unaligned group selection problem with dynamic programming. Our technique also generates balanced sparse networks to get load balance at parallel computing units. Experiments demonstrate that the 2D unaligned group-level pruning shows 3.12% a lower error rate at ResNet-20 network on CIFAR-10 compared to the previous 2D aligned group-level pruning under 95% of sparsity.**

***Index Terms*—Deep Neural Networks, Pruning, DNN compression, Alignment, Cache-aware**

I. Introduction

Deep neural networks (DNNs) have achieved remarkable success in a wide variety of problems such as natural language processing [1], speech recognition [2], and computer vision [3]. To attain high accuracy on complex problems, DNN models are becoming larger and deeper. Therefore, recent DNN models require a tremendous computing cost and energy consumption for the model execution. In particular, for on-device machine learning, it is challenging to run those large DNN models at mobile devices that have limited computing, memory, and power resources.

To resolve such a problem, network pruning techniques [4]–[6] were proposed to remove redundant connections in a neural network. The network pruning can improve inference performance by reducing the number of required arithmetic operations and memory accesses. Recent pruning techniques can be divided into two categories, i.e., fine-grained pruning and coarse-grained pruning. The fine-grained pruning eliminates individual weight connections if they have a small influence on accuracy. Although the pruning can reduce the number of required operations at model execution, it is difficult to utilize hardware parallelism and cache memory with the irregular and unstructured pruned networks, as shown in Fig. 1(a). Moreover, we need a specialized sparse BLAS (Basic Linear Algebra Subprograms) library such as cuSPARSE [7] and clSPARSE [8] to execute sparse model. In contrast, the coarse-grained pruning [9]–[11] removes a whole filter or channel if it has a low impact on accuracy, as shown in Fig. 1(d). Therefore, it generates a structured sparse network that can be easily accelerated by general computing systems. However, the coarse-grained pruning can result in significant accuracy loss when a network is highly compressed.

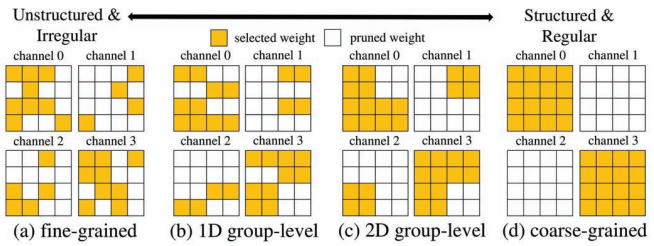


Fig. 1. **Weight matrices under different granularity of network pruning.**

To mitigate the accuracy loss of coarse-grained pruning, several recent approaches [12], [13] proposed *group-level pruning* techniques, where the group size is smaller than the size of filter or channel. If the group dimension (1D or 2D) and size are carefully selected, the group-level pruning can achieve high performance without significant accuracy loss. However, previous group-level pruning techniques used an *aligned* approach, as exemplified in Fig. 1(b) and (c). A given weight matrix is first partitioned into multiple non-overlapping groups, and each group is determined to be pruned or not, based on the total magnitude of the weights in the group. Although this method can easily find victim groups with low complexity of the algorithm, important weights can be removed if their adjacent weights are insignificant. Therefore, the aligned group-level pruning leads to accuracy loss when the network is compressed with a high sparsity. In particular, when the group size is large, the accuracy loss by the aligned approach will be significant.

In this paper, we introduce a flexible pruning approach, called *unaligned* group-level pruning, which can select victim groups without alignment constraint. The weight groups can be selected from arbitrary locations to achieve higher accuracy. However, such an approach can generate more irregular networks compared to the aligned approach. To minimize

978-3-9819263-4-7/DATE20/©2020 EDAA

the performance loss of the unaligned technique, we use a *cache-aware* group selection algorithm and thus there is no significant performance loss. Our specific contributions are as follows:

1) We propose the unaligned group-level pruning algorithm based on dynamic programming. The algorithm considers the hardware factors such as cache line size to prevent performance degradation by the unaligned approach.
2) We propose a balanced pruning technique for the load balance at parallel computing units.
3) Whereas many previous studies reported only the reduction of weight parameters or arithmetic operations by network pruning, we demonstrate the performance improvement by the proposed technique at a real ARM-based mobile device.

II. Related Work

Han *et al.* [4], [5] proposed DNN compression techniques using the fine-grained pruning. This approach first trains neural network connectivity via a normal network training scheme. Next, all connection with weights below a threshold is removed from the network. Finally, it retrains the compressed network to learn the final weights for the remaining sparse connections. The pruning technique reduced the number of parameters by up to 90% for AlexNet and VGG-16 model without accuracy loss. However, it is hard to attain performance gains at real hardware platforms due to the irregular memory access associated with the scattered weight distribution [14]. In addition, it is hard to allocate the same amount of tasks to parallel units. This load imbalance issue can derive crucial effects.

To solve the problem of irregularity in compressed networks, other studies focused on structured sparsity via coarse-grained pruning such as vector-level (1D), kernel-level (2D) [9], and filter-level (3D) [10] pruning. For a given pre-trained dense model, the coarse-grained pruning method uses the magnitude-based pruning based on the sum of the absolute values in groups (L1 norm). As another approach to training a sparse network from scratch, Wen *et al.* [11] proposed a structured sparsity learning method. The group Lasso regularization is used during the training. As a result, they produced structured and regular compressed networks. These coarse-grained pruning approaches bring more regular sparsity patterns, making it easier to leverage hardware acceleration [13]. However, it is challenging to get a high sparsity while maintaining the same accuracy in the coarse-grained pruning.

Considering the trade-off between accuracy and performance at the different granularity of pruning methods, we can consider a medium-grained granularity of pruning, i.e., group-level approach, which prunes adjacent weight elements as a group. The group size is smaller than the filter or channel size. For example, Yu *et al.* [12] proposed a SIMD-aware group-level pruning to utilize the underlying data-parallel SIMD (single instruction multiple data) hardware structure. The group size is configured to be equal to the SIMD width. Therefore, the SIMD hardware units can be fully utilized. However, the accuracy of pruned networks can be limited due to the aligned pruning constraint, as explained in Section 1. Our approach can choose the groups to remain without the alignment constraint, and thus can provide a higher accuracy compared to the aligned schemes under the same sparsity.

III. Unaligned Group-Level Pruning

A. Group Size Determination

The weight group size must be determined considering underlying hardware factors. We must know how many MAC (multiply-and-accumulate) operations can be executed concurrently at parallel computing units. For example, ARM Mali-T628 GPU is scalable up to 8 shader cores. Each shader core has two arithmetic pipelines and one load/store pipeline. Each arithmetic pipeline has four 128-bit SIMD units. Therefore, four 32-bit floating-point operations can be executed simultaneously by each SIMD unit. To make full use of the SIMD unit, the group size must be a multiple of four. In addition, since the cache line size is 64 bytes, the group size must be a divisor of 16. Otherwise, one group may be located across a cache line boundary, resulting in performance degradation. In our experiments, considering the above hardware factors of ARM Mali-T628 GPU, the 1D group size is configured to 4, and the width and height of 2D group are also configured to 4 (4×4).

B. Unaligned Group Selection

Our group-level pruning determines whether a group is important or not based on the magnitude of the group, i.e., the sum of all the weight values in the group[1]. If a group has a smaller magnitude than other groups, the group is regarded as an unimportant group. For example, when the target sparsity is 66.6%, the element-level pruning will generate an irregular sparse matrix as shown in Fig. 2(a). To satisfy the target sparsity, the top 12 elements are preserved (colored elements). Then, the unselected weights are masked to zeros. The sum of preserved weights ($\sum w$) is 102. The legacy aligned group-level pruning can yield a more regular pruned matrix as shown in Fig. 2(b). The 1D group size is 2. Only the groups with magnitudes of higher than 12 are selected, and each group location is aligned to the group size. Due to the aligned group-level approach, the sum of the remaining weights is reduced to 92. In particular, if we compare two matrices in Fig. 2(a) and (b), we can know that several large magnitude of weights are eliminated by the aligned group-level pruning.

If we remove the alignment constraint, we can select weight groups at arbitrary locations and thus can preserve more number of significant elements. Fig. 2(c) shows the result of the unaligned group-level pruning. Under the same sparsity used in Fig. 2(b), the sum of preserved weights is increased. (Although the difference on the sum of preserved weight

[1]To calculate the importance of a group, other functions can be used since our technique just needs to know the importance of a group.

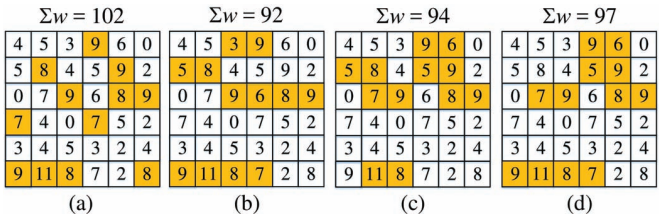


Fig. 2. **Aligned pruning vs. unaligned pruning: (a) element-level, (b) aligned group-level, (c) unaligned group-level (greedy solution), and (d) unaligned group-level (optimal solution)**

values between the aligned and unaligned methods is small in this simple example, the difference can be large in real DNN models as will be demonstrated in Section 5.) As a simple solution for the unaligned group-level pruning, we can consider a greedy algorithm, which selects the largest magnitude of the group from the candidate groups repeatedly as long as the target sparsity is satisfied. The largest magnitude of the group can be located at any location. At each step, not only is the selected group removed from the candidate groups, but other groups that overlap with the selected group are also excluded from the candidate. For example, in Fig. 2(c), the group of (11,8) is first selected. Then, the additional five groups are selected in the order of (8,9), (7,9), (9,6), (5,9), and (5,8). In the aligned approach, if the width and height of the kernel matrix are not a multiple of the group size, several zero values must be padded at the end of each row and column, or the remaining values must be ignored, resulting in accuracy drop. However, the unaligned approach has no such a problem since a group can be selected from any location.

Although the greedy algorithm is simple, the solution is not optimal. In particular, when the matrix size and the group size are large, the greedy algorithm can generate a low quality of the solution. Fig. 2(d) shows the optimal solution at a given sparsity. The total value of preserved weights in the optimal solution is larger than that of the greedy solution.

To design the optimal algorithm, we need to define the unaligned group-level pruning problem. In general, the convolution kernel in CNN is defined as a four-dimensional tensor ($K \times K \times C \times F$). K denotes the size of convolution kernel (i.e., $K \times K$). C and F represent the number of channels and filters, respectively. To make a searching problem in one-dimensional space, the 4-D tensor is transformed into one-dimensional tensor with the size of K^2CF. When n denotes the width of the transformed matrix (i.e., $n = K^2CF$), for a given target sparsity ratio S, the number of groups to be preserved, m, can be calculated, i.e., $m = n \times (1 - S)$. When $W^k_{s,e}$ denotes the total sum of weights in k number of non-overlapping groups selected from the sub-region of the target one-dimensional weight matrix, where the first index and the last index of the sub-region are s and e, respectively, we must find m groups such that $W^m_{1,n}$ is maximized for the optimal solution. If our algorithm satisfies the following equation, we can say that it can find the optimal solution.

$$W^k_{s,e} = \max_{G \le i < 2G} (W^{k-1}_{s,e-i} + \sum_{j=e-i+1}^{e-i+G} |\, w_j \,|\,,\ W^k_{s,e-G}) \quad (1)$$

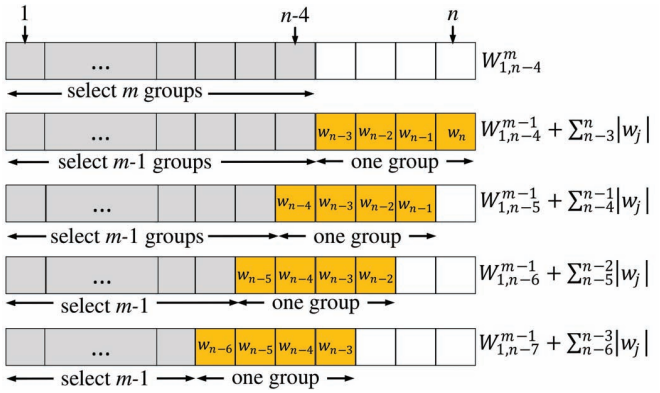


Fig. 3. **The five cases one of which can be a solution to maximize $W^m_{1,n}$ assuming the group size is 4.**

In Equation (1), w_j represents the weight value with the index j and G is the group size. To help the understanding of Equation (1), we show Fig. 5, which illustrates the five cases one of which can be a solution to maximize $W^m_{1,n}$ assuming the group size is 4. Using dynamic programming, the optimal solution satisfying Equation (1) can be found.

After network pruning, the weight value of a pruned connection is changed to zero. Since the zero values are not required at matrix multiplication, the parameters of sparse neural networks are generally stored in a sparse matrix representation such as CSR (Compressed Sparse Row) [15] to reduce storage space and memory access cost. Although the pruned weight matrix is stored at a sparse format, the input feature matrix cannot be compressed in advance and thus it is dense. Therefore, we need the sparse matrix-vector (SpMV) and sparse matrix-matrix multiplication (SpMM) kernels to execute sparse DNNs. The SpMV and SpMM operation load only the input feature values whose corresponding weight values are non-zeros.

When solving the unaligned group selection problem with dynamic programming, we must consider two boundary conditions. First, we must avoid a group that is located across two filters. Since each filter operation uses different output registers, the matrix multiplication with such a boundary-crossing weight group invokes accesses to multiple output registers. Then, the matrix multiplication operation needs an additional *reduce* operation that leads to performance degradation. Second, we must avoid a group that is located across two cache lines in the dense format. This technique is called *cache-aware* unaligned pruning. Fig. 4(a) shows input feature loading operations under different pruning techniques and different group sizes. Since only the input feature values whose corresponding weight values are non-zeros are required, only the colored input values are loaded into cache lines. Since the cache line size is a multiple of group size, there are no input feature groups across the cache line boundary in the aligned pruning. In particular, when the group size is the same as the cache line size, the cache loading operation is optimized. However, the weight matrix compressed by the unaligned pruning can increase the number of cache line loads for input features since some input groups are located across the cache

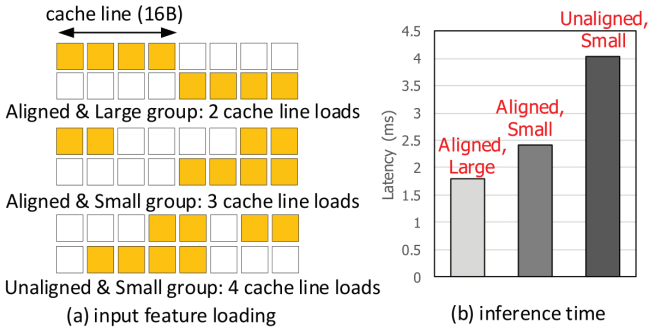


Fig. 4. **Effect of unaligned pruning: the cache line size is 4×4B and the group sizes are 4×4B (large) and 2×4B (small).**

line boundary. Fig. 4(b) shows the execution times of the 16-th convolution layer of the ResNet-20 [16] under different group sizes and alignment policies. We measured the execution times at an ARM-based ODROID-XU4 device and the sparsity of the weight matrix is 88.5%. The *cache-unaware* unaligned approach results in about 2.25 ms of performance degradation compared to the aligned and large group approach. Therefore, we need to use the cache-aware algorithm to minimize the performance degradation by the unaligned pruning.

C. Balanced Sparsity

The SpMV or SpMM operation on a sparse weight matrix is usually computed by multiple threads, each of which processes one row. If all the rows have a similar number of non-zeros, we can achieve better performance due to the balanced loads on multiple threads. Therefore, network pruning needs to generate a balanced sparse matrix. However, if all the rows are restricted to have the same number of weights groups for load balance, the accuracy will be lowered. To consider this trade-off, we use a load balancing factor, B $(0 \leq B \leq 1)$, that represents the intensity of load balance. The load balancing factor determines the lower bound of the sparsity of each row as follows:

$$S_{l,r} \geq S_l * B \tag{2}$$

where $S_{l,r}$ represents the sparsity of the r-th row in the l-th layer of the target neural network, and S_l represents the target sparsity of the l-th layer. Through Equation (2), the maximum number of groups that can be preserved at each row can be determined. For example, if B is 0 as shown in Fig. 6(a), there is no restriction on the number of groups that can be preserved at each row. That is, the load balancing requirement is ignored. Therefore, an accuracy-optimized pruned matrix is obtained. However, if B is 1 as shown in Fig. 6(c), the group-level pruning selects the same number of groups from each row while satisfying the target sparsity of the layer. In Fig. 6(b), since B is 0.67, each row can have a sparsity higher than 41.2% (= 61.6% × 0.67). Therefore, the group-level pruning can preserve up to two groups at each row.

By setting the degree of freedom of the number of groups each row can have, the trade-off between accuracy and performance can be controlled. The load balancing factor is used during the optimal group-level pruning. The dynamic programming excludes a solution from the search space if there is any row that violates the constraint of sparsity lower bound in the solution.

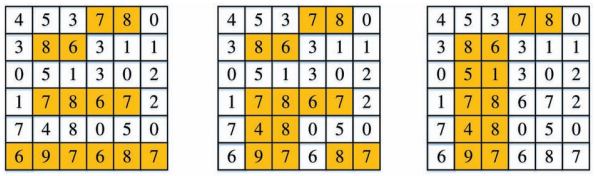


Fig. 5. **Pruned matrices under different load balancing factors.**

IV. Experiments

A. Experimental Environment

Our target hardware platform is ODROID-XU4 board, which is equipped with Exynos 5422 Cortex-A15 2Ghz and Cortex-A7 octa-core CPUs, ARM Mali-T628 GPU, and 2 GB of LPDDR3 RAM. The Mali-T628 GPU has six compute units each of which provides 17 GFLOPS performance. In ODROID-XU4, the six compute units are separated into two devices with four and two units. We used only the device having four compute units in the experiments.

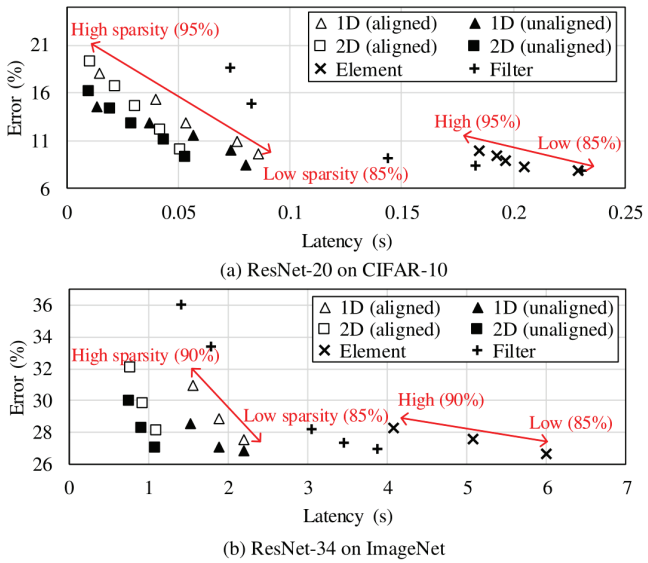
We used ACL (Arm Compute Library) [17] (version 19.02) as a baseline. It has a set of functions optimized for both ARM CPUs and GPUs. For comparison, dense DNN models are executed using ACL. Since ACL does not support an optimized sparse BLAS library, we implemented our own sparse library to execute the sparse DNN models. The experiment shows that our sparse BLAS library outperforms the original ACL.

We implemented our unaligned group-level pruning method in PyTorch. After the group size and the load balancing factor are determined, the cache-aware unaligned groups are selected through a dynamic programming algorithm. The weight values excluding the selected groups are masked to zero, and the final model is obtained through retraining. We evaluate our technique with several popular DNN models: ResNet-20 and ResNet-34 [16]. The network pruning was not applied to the first layer and the last layer for all the networks since these layers are highly sensitive to the removal of weight connections. In all experiments, the 1D group size is configured to 4, and the 2D group size is configured to 4×4. The DNN model executions are accelerated with the Mali GPU.

B. Accuracy Improvement

We first evaluated the accuracy improvement by our technique with the ResNet-20 and ResNet-34 models, each of which used CIFAR-10 and ImageNet datasets, respectively. The error rates of the original dense networks are 7.59% at ResNet-20/CIFAR-10 and 26.09% (Top-1 error) at ResNet-34/ImageNet. For the unaligned pruning, we used the cache-aware optimal search algorithm without the load balancing constraint (i.e. the load balancing factor B is 0).

Fig. 6 shows the trade-off between accuracy and model execution time (inference latency) under different sparsity values. Fig. 6(a) is the result of ResNet-20 on CIFAR-10. For the group-level pruning and element-level pruning, five different results are shown for each method when the sparsity



(a) ResNet-20 on CIFAR-10

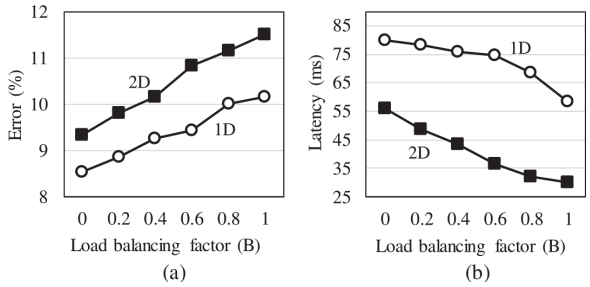
(b) ResNet-34 on ImageNet

Fig. 6. **Latency and accuracy comparison of different pruning techniques (ResNet-20 on CIFAR-10).**

ratios are 85%, 87.5%, 90%, 92%, and 95%. For the filter-level pruning [18], the sparsity ratios of 10%, 20%, 30%, 60%, and 70% were used to show the results at similar error rates or similar latencies. The element-level pruning provides reasonable accuracy at high sparsity but shows about four times lower performance than the group-level pruning owing to the hardware-unfriendly irregular sparse network. The filter-level pruning must use a lower sparsity (i.e, cannot be compressed highly) to provide a similar accuracy as the group-level pruning and thus shows longer latency times even though it creates hardware-friendly structured networks. On the other hand, the filter-level pruning must be compressed with a high sparsity (60% or 70%) to get a similar latency as the group-level pruning, then its error rate increases significantly. Therefore, we can know that both the fine-grained and the coarse-grained pruning are not suitable to compress a DNN model targeting for mobile devices.

The unaligned pruning shows lower error rates than the aligned pruning at the same sparsity. In particular, the difference is more significant in higher sparsity. For example, with the sparsity of 95%, the 1D and 2D unaligned pruning techniques achieve about 3.48% and 3.12% of accuracy improvements, respectively, over the aligned pruning. Although the unaligned pruning shows higher accuracy owing to its flexibility, the irregularity of the pruned network is intensified and the cache miss ratio can be increased. However, the performance difference between the two group-level techniques at the same sparsity is negligible due to the cache-aware pruning.

Fig. 6(b) is the result of a larger-scale model and dataset, i.e., ResNet-34 on ImageNet dataset. For the group-level pruning and element-level pruning, three different results are shown for each method when the sparsity ratios are 85%, 87.5%, and 90%. The sparsity ratios in the filter-level pruning are 10%, 20%, 30%, 60%, and 70%. At 90% sparsity, the 2D unaligned sparse network has an error rate of 29.98%,



(a) (b)

Fig. 7. **Accuracy and latency changes on different load balancing factors (ResNet-20 on CIFAR-10).**

which is 2.04% lower than the 2D aligned sparse network. Similarly, the 1D unaligned sparse network has a 2.4% lower error rate than the 1D aligned sparse network. In particular, the 2D unaligned pruning at 90% of sparsity shows a similar error rate as the 2D aligned pruning at 87.5% of sparsity.

The 85% of sparse network pruned by the element-level pruning requires about 6 seconds to execute at ODROID-XU4, which is 2.7 times and 5.4 times slower than the results of 1D and 2D group-level pruning techniques, respectively. Compared with the group-level pruning, the element-level pruning shows a significant difference in the inference time at different sparsity values. This is because the irregularity of the pruned network can be mitigated at a high sparsity. However, the accuracy will plummet at a too high sparsity.

C. Load balancing

To observe the effect of the load balancing factor, we measured the accuracy and performance changes while varying the load-balancing factor. Fig. 7 shows the trade-off between latency and accuracy under different load-balancing factor values when ResNet-20 is pruned by the 1D and 2D unaligned pruning techniques targeting 85% of sparsity. When the load-balancing factor is 0, the maximum degree of freedom is given to the number of weight groups that can be assigned to each row. As the load-balancing factor increases, the accuracy decreases. As a result, the fully balanced sparse network has 1.64% and 2.18% larger error rates compared to the imbalanced sparse network on 1D and 2D unaligned pruning techniques, respectively. However, even when the sparse network pruned by the unaligned technique is perfectly balanced (i.e., the load-balancing factor is 1), it shows a lower error rate than the unbalanced sparse network pruned by the aligned technique. From this result, we can know that the load balancing technique can improve performance maintaining the benefit of the unaligned group pruning.

D. Greedy vs. Optimal Search Algorithms

Table I shows how much the error rate can be reduced by the optimal unaligned group search algorithm compared with the greedy algorithm on ResNet-20/CIFAR-10. The optimal unaligned pruning algorithm shows about 1∼2% and 1.4∼2.2% better accuracy values than the greedy algorithm at the 1D pruning and 2D pruning, respectively. As the sparsity is higher, the difference between two techniques becomes larger.

TABLE I
ERROR RATES AT DIFFERENT UNALIGNED PRUNING ALGORITHMS

Sparsity	1D pruning		2D pruning	
	greedy	optimal	greedy	optimal
90%	12.58	11.59	14.2	12.8
92.5%	14.49	12.87	16.21	14.32
95%	16.62	14.63	18.43	16.19

From this result, we can say that the proposed optimal pruning algorithm is superior to the greedy unaligned algorithm.

E. Effect of Cache-Aware Pruning

To analyze the effect of the cache-aware unaligned pruning, we evaluated the accuracy and performance difference by the cache-aware technique. As shown in Fig. 8(a), the error rate slightly increases by the cache-aware technique since the cache line boundary condition in the cache-aware technique can exclude the best solution that can be found by the cache-unaware algorithm from the candidates. For example, with the sparsity of 90%, the 1D and 2D cache-aware unaligned pruning techniques increase about 0.34% and 0.46% of error, respectively, compared to the cache-unaware pruning.

However, the cache-aware approach can reduce the latency significantly as shown in Fig. 8(b). The latency improvement by the cache-aware technique is about 8∼17% and 10∼13% at the 1D pruning and 2D pruning, respectively. In addition, we also observed that the performance difference between the two approaches becomes significant when the network is larger and deeper.

V. CONCLUSION

Since normal deep neural network models have a large amount of redundancy, the network pruning is an effective technique to reduce the model size and computation cost without significant accuracy loss. We proposed a novel pruning technique, called unaligned group-level pruning. Unlike the previous group-level pruning, our scheme can select weight groups to be preserved without alignment constraint. Due to its flexibility, the unaligned group sparse networks can provide higher accuracy than the aligned sparse networks. In addition, balanced sparse networks can be generated to efficiently utilize underlying parallel hardware. Based on experimental results, we can conclude that the unaligned group-level pruning can enable the on-device machine learning at resource-limited mobile systems.

ACKNOWLEDGMENT

This work was supported by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIT) (No.IITP-2017-0-00914, Software Framework for Intelligent IoT Devices)

REFERENCES

[1] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proceedings of the 25th international conference on Machine learning*, 2008, pp. 160–167.

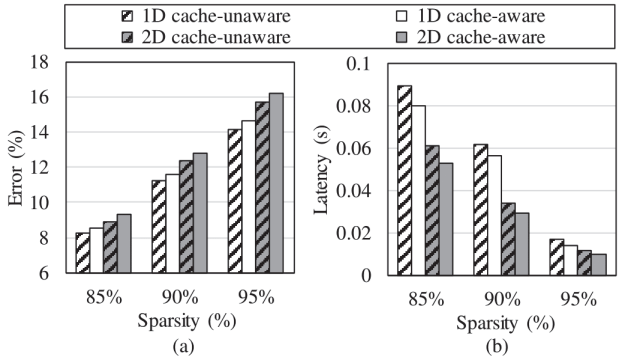


Fig. 8. **Latency and accuracy comparison between two different pruning approaches (ResNet-20 on CIFAR-10).**

[2] D. Amodei, S. Ananthanarayanan, R. Anubhai, J. Bai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, Q. Cheng, G. Chen *et al.*, "Deep speech 2: End-to-end speech recognition in english and mandarin," in *Proceedings of International conference on machine learning*, 2016, pp. 173–182.

[3] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *Proceedings of European Conference on Computer Vision*, 2016, pp. 21–37.

[4] 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.

[5] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Proceedings of the advances in neural information processing systems*, 2015, pp. 1135–1143.

[6] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 2736–2744.

[7] C. Nvidia, "Cusparse library," *NVIDIA Corporation, Santa Clara, California*, 2014.

[8] J. L. Greathouse, K. Knox, J. Poła, K. Varaganti, and M. Daga, "clsparse: A vendor-optimized open-source sparse blas library," in *Proceedings of the 4th International Workshop on OpenCL*, 2016, p. 7.

[9] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 1389–1397.

[10] J.-H. Luo, J. Wu, and W. Lin, "Thinet: A filter level pruning method for deep neural network compression," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 5058–5066.

[11] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Proceedings of the advances in neural information processing systems*, 2016, pp. 2074–2082.

[12] J. Yu, A. Lukefahr, D. Palframan, G. Dasika, R. Das, and S. Mahlke, "Scalpel: Customizing dnn pruning to the underlying hardware parallelism," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2017, pp. 548–560.

[13] H. Mao, S. Han, J. Pool, W. Li, X. Liu, Y. Wang, and W. J. Dally, "Exploring the granularity of sparsity in convolutional neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2017, pp. 13–20.

[14] 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," in *Proceedings of the 43rd Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2016, pp. 243–254.

[15] M. Steinberger, R. Zayer, and H.-P. Seidel, "Globally homogeneous, locally adaptive sparse matrix-vector multiplication on the gpu," in *Proceedings of the International Conference on Supercomputing*. ACM, 2017, p. 13.

[16] 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.

[17] "ARM Compute Library," https://github.com/ARM-software/ ComputeLibrary.

[18] Y. He, G. Kang, X. Dong, Y. Fu, and Y. Yang, "Soft filter pruning for accelerating deep convolutional neural networks," in *IJCAI International Joint Conference on Artificial Intelligence*, 2018.