This CVPR workshop paper is the Open Access version, provided by the Computer Vision Foundation.
Except for this watermark, it is identical to the version available on IEEE Xplore.

Exploring the Granularity of Sparsity in Convolutional Neural Networks

Huizi Mao[1], Song Han[1], Jeff Pool[2], Wenshuo Li[3], Xingyu Liu[1], Yu Wang[3], William J. Dally[1,2]
[1]Stanford University
[2]NVIDIA
[3]Tsinghua University
{huizi,dally}@stanford.edu

Abstract

Sparsity helps reducing the computation complexity of DNNs by skipping the multiplication with zeros. The granularity of sparsity affects the efficiency of hardware architecture and the prediction accuracy. In this paper we quantitatively measure the accuracy-sparsity relationship with different granularity. Coarse-grained sparsity brings more regular sparsity pattern, making it easier for hardware acceleration, and our experimental results show that coarse-grained sparsity have very small impact on the sparsity ratio given no loss of accuracy. Moreover, due to the index saving effect, coarse-grained sparsity is able to obtain similar or even better compression rates than fine-grained sparsity at the same accuracy threshold. Our analysis, which is based on the framework of a recent sparse convolutional neural network (SCNN) accelerator, further demonstrates that it saves $30\% - 35\%$ *of memory references compared with fine-grained sparsity.*

1. Introduction

Deep Neural Networks (DNNs) have many parameters, which leads to problems related to storage, computation and energy cost. State-of-art Convolutional Neural Network (CNN) models have hundreds of millions parameters and take tens of billions operations[12, 16, 23]. That makes DNN models difficult to deploy on embedded systems with limited resources.

To deal with this problem, various methods have been proposed to compress DNN models and reduce the amount of computation. Some methods are based on decomposition and reduction[27, 17]. They usually preserve the regularity of the original models, thus are able to to achieve both compression and acceleration on general-purpose processors. Pruning serves as another effective method to greatly reduce the number of parameters with almost no loss of accuracy[11, 7].

Pruning based methods are generally better at preserving

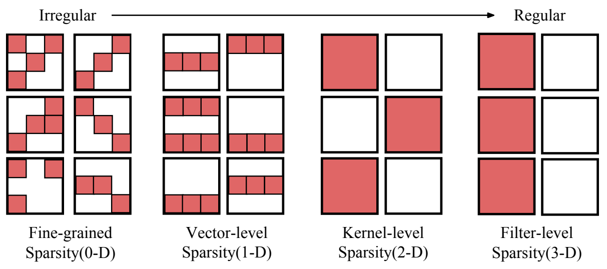
Irregular
Regular
Fine-grained Sparsity(0-D)
Vector-level Sparsity(1-D)
Kernel-level Sparsity(2-D)
Filter-level Sparsity(3-D)

Figure 1. Different structure of sparsity in a 4-dimensional weight tensor. Regular sparsity makes hardware acceleration easier.

accuracy as well as achieving higher compression rates[11]. However, such improvements come at the cost of regularity. Moreover, it has been shown that pruning channels will cause larger accuracy loss than pruning individual weights[20]. Those observations pose several questions: *What is the trade-off between regularity and accuracy? Is it possible to find a sweet spot in the range of regularity?*

We attempt to answer those questions by looking into pruning with different granularity. The structure of sparsity, as shown in Figure 1, not only impacts the prediction accuracy but also affects the efficiency of hardware architecture. There are already some existing works trying to prune filters or channels instead of individual weights[21, 26, 1]. However, due to the various methods they used, we cannot directly evaluate the relationship between pruning granularity and final accuracy. We therefore apply the exact method and experimental setting for an effective comparison. We also want to explore a consistent range of granularity, which includes intermediate grain size like kernels and sub-kernel vectors. Based on a thorough space exploration, we are able to analyze the storage saving and hardware efficiency at different granularity of sparsity.

In this work, we make the following contributions:

- We explore a complete range of pruning granularity and evaluate how it affects the prediction accuracy.

- We demonstrate that coarse-grained pruning is able to reach similar or even better compression ratio than the fine-grained one, even though it obtains less sparsity.
- We show that coarse-grained sparsity is more hardware-friendly and more energy-efficient for sparse neural network accelerators.

2. Related Works

Sparsity has proven to be an effective approach to save parameters of Deep Neural Network models[11, 7]. A number of works investigate how to select the important connections and effectively recover the accuracy. Second-order derivative[19], absolute value[11], loss-approximating Taylor expansion[21], and output sensitivity[5] are examples of importance metrics used for pruning. There are also methods trying to better integrate pruning and training, like iterative pruning[11] and dynamic pruning[7].

The sparsity caused by network pruning typically results in an irregular workload, which is difficult for acceleration. Recent work tries to alleviate this problem by enforcing *structured* sparsity. It tries to limit sparsity to the high-level structure of tensors. Most of the works are targeted at Filter pruning and Channel pruning[26, 20, 21]. Finer-grained structured sparsity is also studied, including intra-kernel strided pruning [1].

For very coarse-grained sparsity like Filter-sparsity and Channel-sparsity, it is simple to achieve acceleration on general-purpose processors because it is equivalent to obtaining a smaller dense model[26]. For fine-grained sparsity, custom accelerators[9, 22] have been used to exploit the reduction of computations.

3. Granularity of Sparsity

3.1. Notations

To simplify the description, we use the following notations for CNN. In a single convolutional layer, the weights compose a 4-dimensional tensor of shape $C \times K \times R \times S$. C is the output dimension, i.e., the number of output feature maps. K is the input dimension. R and S are the shape of convolution kernels.

One layer's weights consists of multiple filters(3-dimensional tensor of shape $K \times R \times S$), each one associated with an output feature map. The weights can also be viewed as multiple channels(3-dimensional tensor $C \times R \times S$), each one associated with an input feature map. Filters and channels are both composed of kernels(2-dimensional tensor $R \times S$) which are the key element in the 2-d convolution operation. Sub-kernel vectors (1-dimensional tensor of size R or S) and scalar weights(0-dimensional tensor) are lower-level elements in a conv layer. Figure 2 illustrates these concepts.

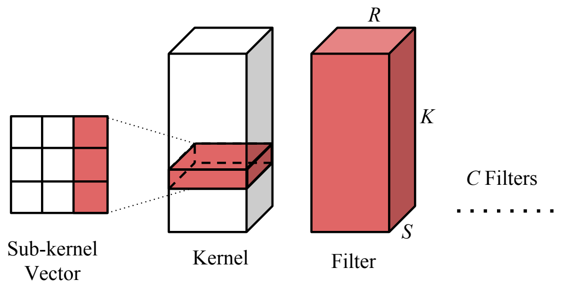

Figure 2. Illustration of the concepts of Filter, Kernel and Sub-kernel vector.

3.2. Range of Granularity

Sparsity in Deep Neural Network, explicit or implicit, have been studied in a lot of literature. Among all types of sparsity, vanilla sparsity(fine-grained sparsity) and filter-wise sparsity(very coarse-grained sparsity) are two extreme cases that are widely studied[11, 20].

Vanilla sparsity is a type of sparsity in which individual weights are masked as zero, first proposed in 1989 by LeCun et al.[19]. The fine-grained sparsity has been proven to work well on a wide range of popular neural network models of CNN, RNN and LSTM [11, 7, 6, 8].

There is also implicit sparsity used for neural network. Channel reduction is a type of neural network compression technique which reduces the dimension of input/output features and thereby reduces the size of a layer. Channel reduction can be viewed as very coarse-grained sparsity, which removes 3-dimensional sub-tensors in Convolutional layers and 1-dimensional vectors in Fully-Connected layers. Such coarse-grained sparsity is beneficial for acceleration due to regularity[18, 26]. However, it usually causes noticeable accuracy drops compared with fine-grained sparsity, as indicated by Li et al.[20].

There is a large range of granularity between vanilla sparsity and channel reduction. Some literature attempts to explore one or a few possibilities among all choices. Channel pruning and intra-kernel strided pruning have been investigated in the work of Anwar et al.[1]. However, compared with the large range of possible grain sizes, the attempts so far are still incomplete.

In this paper, we investigated two types of granularity ranges. The first one is dimension-level granularity in which the grain size increases with the number of dimensions. We explore 4 granularities. To be specific, we study the accuracy-sparsity relationship when the atomic elements(grain) during pruning are filters, kernels, sub-kernel vectors or scalar weights. The other type is stride-level granularity, in which the grains have the same number of dimensions but vary in the sizes. For example, the pruned grains can be size-2 vectors, size-8 vectors or size-C vec-

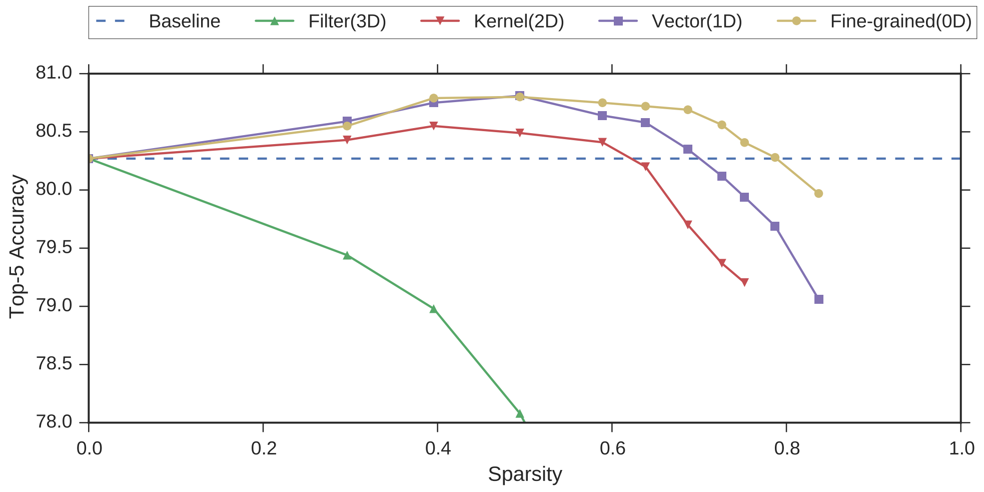

Figure 3. Accuracy-Sparsity Curve of AlexNet with different grain sizes. X-axis: sparsity of conv layers (percentage of zero weights). Y-axis: top-5 accuracy on ImageNet validation set.

tors. Here we only study the simplified case where the pruned grains are all 1-dimensional vectors lying along C dimension(the output dimension).

We explain these two types with numpy-like codes as below. For dimension-level granularity, Figure 1 also illustrates the dimension-level granularity of sparsity.

```
 Weights = Array(C, K, R, S)

# Case 1: Dimension−level granularity
 Filter(3−Dim) = Weights[c, :, :, :]
 Kernel(2−Dim) = Weights[c, k, :, :]
 Vector(1−Dim) = Weights[c, k, r, :]
 Fine−grain(0−Dim) = Weights[c, k, r, s]

# Case 2: Stride−level granularity
 Vector_stride_i = Weights[ci:(c+1)i,k,r,s]
 Vector_stride_C = Weights[:, k, r, s]
```

3.3. Coarse-grained Pruning Method

Coarse-grained pruning deletes multiple weights together instead of individual weights. Typically the grain can be filters, kernels, sub-kernels vectors or anything else. Because we are interested in the effects of the grain size rather than the pruning method, we adopt the simple magnitude-based pruning criterion in [11]. For a grain G_i that consist of multiple weights, the Salience S_i is defined as the sum of absolute values, as $S_i = \sum_{w \in G_i} |w|$, i.e. based on the L1 norm. Given the targeted sparsity, the grains with small L1 norm are deleted.

We also adopt the iterative pruning method proposed by Han et al.[11]. It is able to reach higher sparsity than direct pruning. The sparsity during each pruning stage is determined by sensitivity analysis, which requires individually pruning every layer and measure the accuracy loss on the training dataset.

4. Accuracy-Sparsity Relation with Different Grain Sizes

Our goal is to study how the granularity of pruning influences the accuracy. Specifically, we want to compare the accuracy of different sparsity structure at the same sparsity . The intuition is that more irregular sparsity pattern, i.e., smaller grain size, usually leads to higher accuracy at the same sparsity.

To ensure fair comparison, we enforce the identical sparsity setting and training schedule for the same model. All experiments were performed on ImageNet dataset[4] with Caffe[14].

For CNN models, we only count the overall sparsity of convolutional layers. One important reason is that there is no such range of granularity for fully-connected layers. Also, convolutional layers take up most of the computations in a typical CNN model[2]. However, we still prune the fully-connected layers(fine-grained pruning for dimension-level granularity, coarse-grained pruning for stride-level granularity) together with convolutional layers, to obtain consistent comparisons with previous works[11, 7].

4.1. Dimension-level granularity

We selected AlexNet for detailed accuracy-sparsity curves. For other networks, VGG-16, GoogLeNet[24], ResNet-50[12], and DenseNet-121 [13]) we also compared the same-sparsity accuracies of different pruning granularity. Their results are reported and compared in Table 1 .

Figure 3 shows the accuracy curve of density(one minus sparsity) under various settings. In this figure there are four

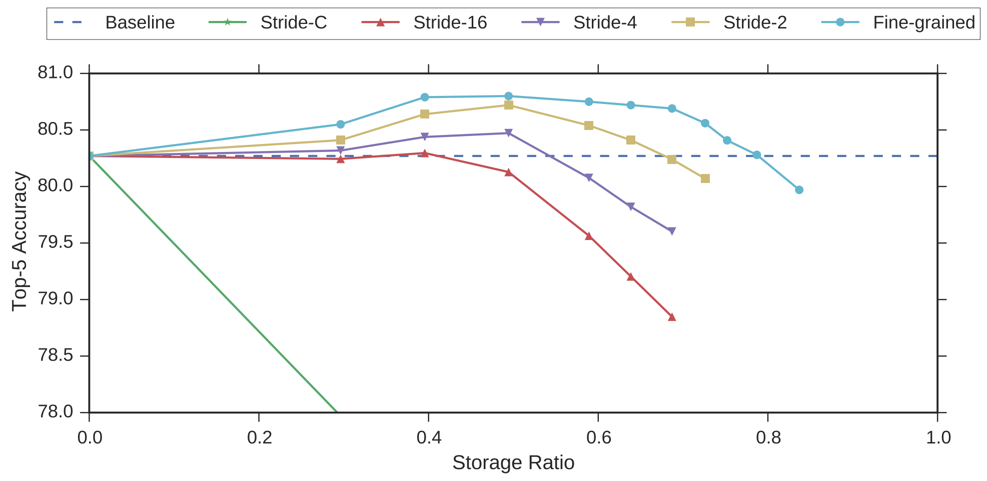

Figure 4. Accuracy-Sparsity Curve of AlexNet with different pruning dimensions.

different granularity of sparsity, in each case the atomic element for pruning is

- **Fine-grained(0-Dim)**: Individual weights.
- **Vector(1-Dim)** : Sub-kernel vectors of size S.
- **Kernel(2-Dim)**: Kernels of shape $R \times S$.
- **Filter(3-Dim)**: Filters of shape $K \times R \times S$.

When the grain size of pruning is very large, say, a filter, we notice huge accuracy loss during pruning. AlexNet loses nearly 1% validation accuracy at the very first pruning stage. For finer-grained ones, the accuracy loss is much smaller and we even noticed small accuracy increases during the first several pruning stages. Note that the results of AlexNet are better than the original work by Han et al.[11]. We give a detailed description in Table 3.

4.2. Stride-level granularity

Figure 4 shows the accuracy-density curve with different pruning strides. In this figure there are five different stride sizes: 1, 2, 4, 16, C. Notice that fully-connected layers are also pruned with the stride. When the stride size is 1, it is equivalent to fine-grained pruning. Here we observed the similar results that accuracy drops when increasing the grain size. Deleting vectors along the whole dimension C leads to great accuracy loss.

4.3. Discussion

Our intuition is that sparsity serves as the regularization because it lowers the capacity of the model by reducing the number of parameters. Coarse-grained sparsity works as strong regularization, as it not only reduces the number of parameters but also constrain the positions of parameters. The experimental results support such an assumption.

Table 1. Comparison of accuracies with the same density of convolutional layers. For a given density, fine-grained pruning gives the highest accuracy.

Model	Density	Granularity	Top-5
AlexNet	24.8%	Kernel	79.20%
		Vector	79.94%
		Fine-grained	**80.41%**
VGG-16	23.5%	Kernel	89.70%
		Vector	90.48%
		Fine-grained	**90.56%**
GoogLeNet	38.4%	Kernel	88.83%
		Vector	89.11%
		Fine-grained	**89.40%**
ResNet-50	40.0%	Kernel	92.07%
		Vector	92.26%
		Fine-grained	**92.34%**
DenseNet-121	30.1%	Kernel	91.56%
		Vector	91.89%
		Fine-grained	**92.21%**

We also find that pruning with a large grain size, e.g., the size of a filter or output dimension C, will greatly hurt the accuracy. On the other hand, pruning with a smaller grain size leads to an accuracy-sparsity curve similar with fine-grained pruning. Notice that in Figure 3& 4, some curves appear to rise smoothly at first. That suggests coarse-grained pruning can still reach similar compression rates as fine-grained pruning, giving additional advantages that will be described in the following section.

5. Comparison of Storage

Model size is an important factor for real-world applications. It has been pointed out that memory access takes up a large portion of total energy consumption during the exe-

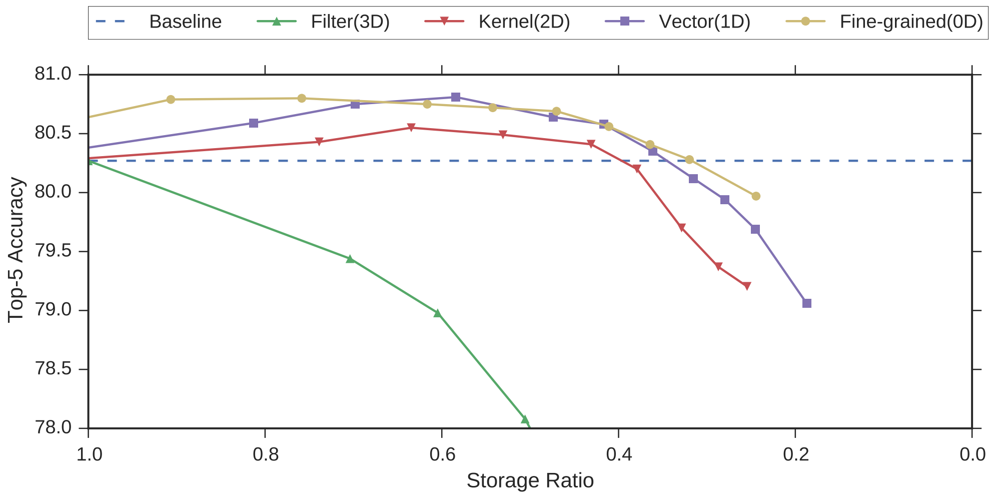

Figure 5. Accuracy-Storage Curve of AlexNet with different grain sizes.

cution of deep neural network[11]. Sparsity is an effective approach to compress neural network models. Sparse neural network is usually stored with a similar format to Compressed Row Storage(CRS) for sparse matrix, where both values and indexes are stored. Coarse-grained sparsity, due to its regularity, is able to save the number of indexes as illustrated in Figure 6. Therefore the coarse-grained sparse models take up less storage than fine-grained ones, when they are at the same level of sparsity.

We want to investigate how accuracies differ at the same level of storage(instead of sparsity) for different granularity of pruning. We do not use full-precision 32-bit weights but use 8-bit weights instead, as 8-bit weights, either true 8-bit integer formats or 8-bit indices into a table of shared fp32 weights, have been proven to be sufficient in a lot of literature[15, 9, 25]. We use 4-bit indices to store the distances between adjacent non-zeros, following the method in Deep Compression [10]. Moreover, as indicated in the Deep Compression paper, the quantization method works independently with sparsity. To validate this claim with coarse-grained sparsity, we plot the accuracy-bits curves of different types of pruned models in Figure 7. The results show that sparsity architecture has negligible effect for even 6-bit quantization (64 unique fp32 weight values).

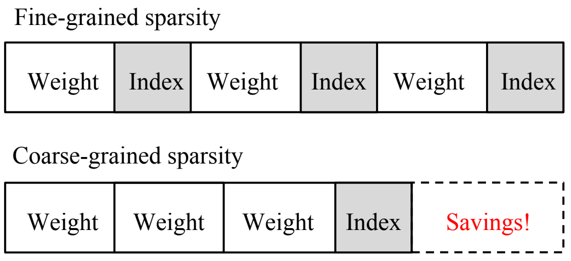

Figure 6. Index saving of coarse-grained sparsity.

Figure 5 shows the accuracy-storage relationship of AlexNet. We find that the first three curves(Fine-grained, Vector and Kernel) are closer than those in Figure 3. It indicates the effects of index saving for coarse-grained pruning.

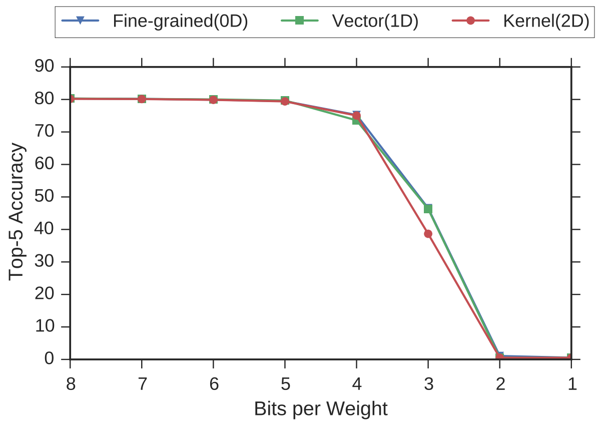

Figure 7. Accuracy-bits curves for sparse AlexNet with different grain sizes. Three curves are almost identical, indicating sparsity structure does not impact quantization.

To better compare the compression ratio under the same accuracy constraint, we list the results of AlexNet, VGG-16 and GoogLeNet in Table 2. Here the storage ratio is defined as the model size of pruned 8-bit model(with 4-bit indices) to that of dense 8-bit model. Notice that it is almost impossible to prune a model that exactly match the baseline accuracy, so we use linear interpolation to obtain the estimated density and storage ratio.

For a sparse network, the larger the grain size is, the

Table 2. Comparison of storage savings at the baseline accuracy(estimated with linear interpolation). Storage ratio is compared with 8-bit dense model.

Model	Top-5 Accuracy	Granularity	Density	Storage Ratio
AlexNet	80.3%	Kernel	37.8%	39.7%
		Vector	29.9%	34.5%
		Fine-grained	22.1%	**33.0%**
VGG-16	90.6%	Kernel	44.4%	46.9%
		Vector	30.7%	**35.8%**
		Fine-grained	27.0%	40.6%
GoogLeNet	89.0%	Kernel	43.7%	51.6%
		Vector	36.9%	**47.4%**
		Fine-grained	32.3%	48.5%
ResNet-50	92.3%	Kernel	61.3%	77.0%
		Vector	40.0%	**52.7%**
		Fine-grained	37.1%	55.7%
DenseNet-121	91.9%	Kernel	35.5%	48.9%
		Vector	31.1%	43.8%
		Fine-grained	26.6%	**39.8%**

less storage it takes. This is due to index sharing among the weights of the kernel as shown in Figure 6. However, AlexNet and VGG-16 in particular have much closer density/storage results for kernel pruning than GoogLeNet and ResNet do. It is caused by the small size of the convolutional kernels being pruned. GoogLeNet and ResNet have 1x1 convolutions in nearly 50% of their layers, which do not get any benefit from sharing index values. AlexNet and VGG-16, on the other hand, have a multitude of larger convolutions.

6. Advantages of Coarse-grained Sparsity

It has been mentioned in the previous sections that filter pruning is able to obtain acceleration on general-purpose processors like CPU or GPU. For intermediate grain sizes like kernels or sub-kernel vectors, though it is still difficult for acceleration on general-purpose processors, there are several advantages over fine-grained sparsity. Those advantages enable simpler circuits and higher energy efficiency on custom hardware. We qualitatively and quantitatively analyze the advantages as follows:

Qualitative analysis. In conv layers, 2-D convolution is usually the primitive operation. Winograd decomposition, which is adopted in recent versions of cuDNN[1], is targeted at reducing computations of 2-D convolution. Kernel pruning can therefore easily map to computation reduction, because the 2-D convolutions of deleted kernels can be deleted as well. Recent custom hardware design for CNN also use 1-D convolution as the primitive operation[3]. In this case, sub-kernel vector pruning is beneficial. Coarse-grained pruning is able to preserve the low-level computation logics, therefore simplify the hardware design.

[1]Nvidia developer's blog

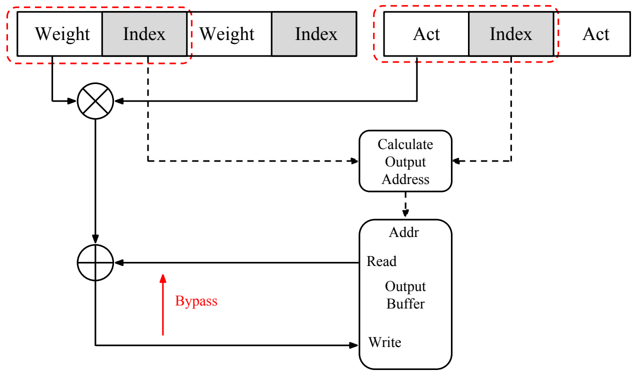

Figure 8. Dataflow of SCNN architecture. Bypass is possible when the same output address is referenced again.

Quantitative analysis. Memory reference is a major factor of energy consumption[11]. Recent work on custom hardware exploits both the sparsity of weights and activations of CNN[22]. In their implementation, the weights and input activations are both stored in sparse format while output activations are stored in dense format. The indices of weights and activations are used for calculating the output address, to which the product of weight and activation will be accumulated. This process is illustrated in Figure8. After one layer is finished, the output activations will then be compressed into the sparse format for next layer.

If the same output address is referenced again, data shortcut can be used to avoid the expensive write/read. For example, two adjacent weights and two adjacent activations will reference 3 addresses instead of 4. Due to the locality, coarse-grained sparse weights have a larger probability

Table 3. Comparison of pruned AlexNet with previous works.

Layer	Param.	NIPS'15 [11]	NIPS'16 [7]	Fine-grained (ours)	Vector Pruning (ours)	Kernel Pruning (ours)
conv1	35K	84%	**54%**	83%	83%	83%
conv2	307K	38%	41%	**26%**	26%	26%
conv3	885K	35%	28%	**23%**	23%	23%
conv4	664K	37%	32%	**23%**	23%	23%
conv5	443K	37%	33%	**23%**	23%	23%
fc6	38M	9%	**3.7%**	7%	7%	7%
fc7	17M	9%	**6.6%**	7%	7%	7%
fc8	4M	25%	**4.6%**	18%	18%	18%
Total	61M	11%	**5.7%**	8.4%	8.4%	8.4%
FLOPs	1.5B	30%	25.4%	**24.1%**	24.1%	24.1%
Storage(conv)	2.3MB	55.6%	48.3%	36.4%	28.0%	**25.5%**
Storage(total)	61MB	16.7%	**8.5%**	12.6%	12.3%	12.2%
Mem Refs	99M	74.4%	71.7%	60.5%	**34.6%**	35.2%
Top-5 Accuracy		80.23%	80.01%	**80.41%**	79.94%	79.20%

Table 4. Comparison output memory references for VGG-16 in units of billion(B) references(only conv layers).

Density	Fine-grained (0-D)	Vector (1-D)	Relative Cost
40.1%	1.77B	1.23B	**69.5%**
33.1%	1.53B	1.03B	**67.2%**
27.5%	1.33B	0.87B	**65.3%**

of output address collision. We simulated with VGG-16 on imageNet validation set to compare the number of memory references of fine-grained sparsity and vector-level sparsity and listed the results in Table 4. It shows that with the same density, coarse-grained sparsity saves $30\% - 35\%$ memory references.

7. Summary

Table 3 gives an overall comparison of key statistics for AlexNet. By using a smoother pruning process, we find the results of Song et al.[11] can be further improved without any algorithmic change. Here FLOPs is the total number of floating-point operations of a model. Storage is measured with the setting of 8-bit weights and 4-bit indexes, as indicated in Section 5. Due to the fact that the storage of conv layers is much smaller but reused much more frequently than fc layers, we add a row for conv layers. The number of memory referenced is calculated by simulating the process of Figure 8.

8. Conclusion

We have explored the granularity of sparsity with experiments on detailed accuracy-density relationship. We pointed out that due to the saving of indices, coarse-grained pruning is able to achieve a similar compression ratio for AlexNet and higher ratios for VGG-16 and GoogLeNet. We also analyze the hardware-level advantages and show that coarse-grained sparsity saves 30% to 35% of memory access. Given the advantages of simplicity and concurrency from hardware perspective, we believe that coarse-grained sparsity can enable more efficient architecture design of Deep Neural Network.

References

[1] S. Anwar, K. Hwang, and W. Sung. Structured pruning of deep convolutional neural networks. *J. Emerg. Technol. Comput. Syst.*, 13(3):32:1–32:18, Feb. 2017.

[2] H. Bagherinezhad, M. Rastegari, and A. Farhadi. Lcnn: Lookup-based convolutional neural network. *arXiv preprint arXiv:1611.06473*, 2016.

[3] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits*, 2016.

[4] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[5] A. P. Engelbrecht. A new pruning heuristic based on variance analysis of sensitivity information. *IEEE transactions on Neural Networks*, 12(6):1386–1399, 2001.

[6] C. L. Giles and C. W. Omlin. Pruning recurrent neural networks for improved generalization performance. *IEEE transactions on neural networks*, 5(5):848–851, 1994.

[7] Y. Guo, A. Yao, and Y. Chen. Dynamic network surgery for efficient dnns. In *Advances In Neural Information Processing Systems*, pages 1379–1387, 2016.

[8] S. Han, J. Kang, H. Mao, Y. Hu, X. Li, Y. Li, D. Xie, H. Luo, S. Yao, Y. Wang, et al. Ese: Efficient speech recognition engine with sparse lstm on fpga. In *Proceedings of*

the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, pages 75–84. ACM, 2017.

[9] 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 International Symposium on Computer Architecture*, pages 243–254. IEEE Press, 2016.

[10] 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.

[11] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, pages 1135–1143, 2015.

[12] 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*, pages 770–778, 2016.

[13] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten. Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993*, 2016.

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

[15] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, and e. a. . In-datacenter performance analysis of a tensor processing unit. In *44th International Symposium on Computer Architecture*, 2017.

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

[17] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *arXiv preprint arXiv:1412.6553*, 2014.

[18] V. Lebedev and V. Lempitsky. Fast convnets using group-wise brain damage. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2554–2564, 2016.

[19] Y. LeCun, J. S. Denker, S. A. Solla, R. E. Howard, and L. D. Jackel. Optimal brain damage. In *NIPs*, volume 2, pages 598–605, 1989.

[20] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

[21] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz. Pruning convolutional neural networks for resource efficient transfer learning. *International Conference on Learning Representations*, 2017.

[22] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. Keckler, and W. J. Dally. Scnn: An accelerator for compressed-sparse convolutional neural networks. In *44th International Symposium on Computer Architecture*, 2017.

[23] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[24] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[25] V. Vanhoucke, A. Senior, and M. Z. Mao. Improving the speed of neural networks on cpus. In *Proc. Deep Learning and Unsupervised Feature Learning NIPS Workshop*, volume 1, page 4. Citeseer, 2011.

[26] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems*, pages 2074–2082, 2016.

[27] X. Zhang, J. Zou, K. He, and J. Sun. Accelerating very deep convolutional networks for classification and detection. *IEEE transactions on pattern analysis and machine intelligence*, 38(10):1943–1955, 2016.