Fast On-the-fly Retraining-free Sparsification of Convolutional Neural Networks [1]

Amir H. Ashouri
University of Toronto
Canada
aashouri@ece.utoronto.ca

Tarek S. Abdelrahman
University of Toronto
Canada
tsa@ece.utoronto.ca

Alwyn Dos Remedios
Qualcomm Inc.
Canada
adosreme@qti.qualcomm.com

Abstract

Modern Convolutional Neural Networks (CNNs) are complex, encompassing millions of parameters. Their deployment exerts computational, storage and energy demands, particularly on embedded platforms. Existing approaches to prune or *sparsify* CNNs require retraining to maintain inference accuracy. Such retraining is not feasible in some contexts. In this paper, we explore the sparsification of CNNs by proposing three model-independent methods. Our methods are applied on-the-fly and require no retraining. We show that the state-of-the-art models' weights can be reduced by up to 73% (compression factor of 3.7×) without incurring more than 5% loss in Top-5 accuracy. Additional fine-tuning gains only 8% in sparsity, which indicates that our fast on-the-fly methods are effective.

1 Introduction

There has been a significant growth in the number of parameters (i.e., layer weights), and the corresponding number of multiply-accumulate operations (MACs), in state-of-the-art CNNs [15, 14, 20, 24, 9, 12, 25, 23]. Thus, it is to no surprise that several techniques exist for "pruning" or "sparsifying" CNNs (i.e., forcing some model weights to 0) to both compress the model and to save computations during inference. Examples of these techniques include: iterative pruning and retraining [3, 8, 4, 21, 18], Huffman coding [6], exploiting granularity [16, 5], structural pruning of network connections [26, 17, 1, 19], and Knowledge Distillation (KD) [10].

A common theme to the aforementioned techniques is that they require a retraining of the model to fine-tune the remaining non-zero weights and maintain inference accuracy. Such retraining, while feasible in some contexts, is not feasible in others, particularly industrial ones. For example, for mobile platforms, a machine learning model is typically embedded within an app for the platform that the user directly downloads. The app utilizes the vendor's platform runtime support (often in the form of a library) to load and use the model. Thus, the platform vendor must sparsify the model at runtime, i.e., *on-the-fly*, within the library with no opportunity to retrain the model. Further, the vendor rarely has access to the labelled data used to train the model. While techniques such as Knowledge Distillation [10] can address this lack of access, it is not possible to apply it on-the-fly.

In this paper, we develop fast retraining-free sparsification methods that can be deployed for on-the-fly sparsification of CNNs in the contexts described above. There is an inherent trade-off between sparsity and inference accuracy. Our goal is to develop *model-independent* methods that result in large sparsity with little loss to inference accuracy. We develop three model-independent sparsification methods: *flat*, *triangular*, and *relative*. We implement these methods in TensorFlow and use the framework to evaluate the sparsification of several pretrained models: Inception-v3, MobileNet-v1, ResNet, VGG, and AlexNet. Our evaluation shows that up to 81% of layer weights in some models may be forced to 0, incurring only a 5% loss in inference accuracy. While the relative method appears to be more effective for some models, the triangular method is more effective for others. Thus, a predictive modeling autotuning [7, 2] is needed to identify, at run-time, the optimal choice of method and it hyper-parameters.

2 Sparsification Methods

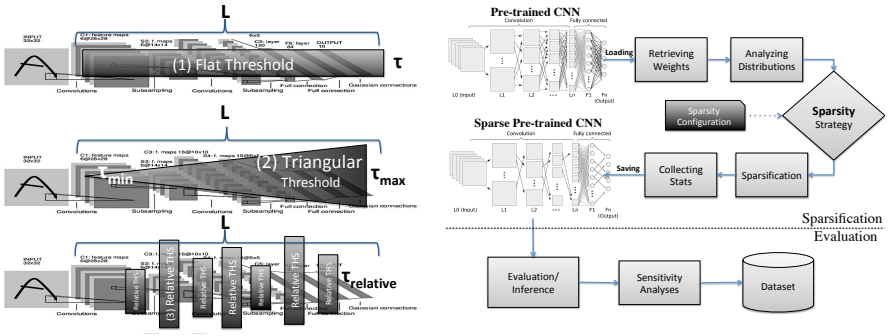
Sparsity in a CNN stems from three main sources: (1) weights within convolution (`Conv`) and fully-connected (`FC`) layers (some of these weights may be zero or may be forced to zero); (2) activations of layers, where the often-applied `ReLU` operation results in many zeros [22]; and (3) input data, which may be sparse. In this paper, we focus on the first source of sparsity, in both `Conv` and fully connected layers. This form of sparsity can be determined a priori, which alleviates the need for specialized hardware accelerators.

The input to our framework is a CNN that has L layers, numbered $1 \ldots L$. The weights of each layer l are denoted by ω_l. We sparsify these weights using a sparsification function, $\mathbb{S}$, which takes as input ω_l and a threshold τ_l from a vector of thresholds T. Each weight i of ω_l is modified by $\mathbb{S}$ as follows:

$$\mathbb{S}(\omega_l(i), \tau_l) = \begin{cases} 0 & \text{if } |\omega_l(i)| \leq \tau_l \\ \omega_l(i) & \text{otherwise} \end{cases} \quad (1)$$

Conference on Neural Information Processing Systems (NIPS), CDNNRIA Workshop, 2018, Montréal, Canada.

[1] The extended version of this paper can be found at: `https://arxiv.org/abs/1811.04199`



(a) Sparsification Methods (b) Sparsification Workflow

Figure 1: Proposed Sparsification Framework

where $\tau_l = T(l)$ is the threshold used for layer l. Thus, applying a single threshold τ_l forces weights within $-\tau_l$ and $+\tau_l$ in value to become 0. Our use of thresholds to sparsify layers is motivated by the fact that recent CNNs' weights are distributed around the value 0.

The choice of the values of the elements of the vector T defines a *sparsification method*. These values impact the resulting sparsity and inference accuracy. We define and compare three sparsification methods. The *flat* method defines a constant threshold τ for all layers, irrespective of the distribution of their corresponding weights. The *triangular* method is inspired by the size variation of layers in some state-of-the-art CNNs, where the early layers have smaller number of parameters than latter layers. Finally, the *relative* method defines a unique threshold for each layer that sparsifies a certain percentage of the weights in the layer. The three methods are depicted graphically in Figure 1a. The high level work-flow of the sparsification framework is depicted in Figure 1b.

Flat Method This method defines a constant threshold τ for all layers, irrespective of the distribution of their corresponding weights. It is graphically depicted in the top of Figure 1a. The weights of the layers are profiled to determine the span $\sigma_{min} = \min_{\forall lin L}(max(\omega_l) - min(\omega_l))$. This span corresponds to the layer k having the smallest range of weights within the pretrained model. This span is used as an upper-bound value for our flat threshold τ. Since using σ_{min} as a threshold eliminates all the weights in layer k and is likely to adversely affect the accuracy of the sparsified model, we use a fraction δ, $0 \leq \delta \leq 1$, of the span $\tau_l = \sigma_{min} \times \delta$, where δ is a parameter of the method that can be varied to achieve different degrees of model sparsity.

Triangular Method The triangular method is defined by two thresholds τ_{min} and τ_{max} for respectively the first convolution layer (i.e, layer 1) and the last fully connected layer (i.e., layer L). They represent the thresholds at the tip and the base of the triangle in middle part of Figure 1a. These thresholds are determined by the span of the weights in each of the two layers. Thus,

$$\tau_l = \begin{cases} \tau_{min} = \sigma_{conv} * \delta_{conv} & \text{if} \quad l = 1 \\ \tau_{max} = \sigma_{fc} * \delta_{fc} & \text{if} \quad l = L \\ \dfrac{\tau_{max} - \tau_{min}}{L} \times (l-2) & \text{Otherwise} \end{cases} \tag{2}$$

where σ_{conv} is the span of the weights in the first convolution layer, defined in a similar way as for the flat method, and it represents an upper bound on τ_{min}. Thus, δ_{conv} is a fraction that ranges between 0 and 1. Similarly, σ_{fc} is the span of the weights in the last fully connected layer and it represents an upper bound on τ_{max}. Thus, δ_{fc} is a fraction that ranges between 0 and 1. The thresholds of the remaining layers are dictated by the position of these layers in the network.

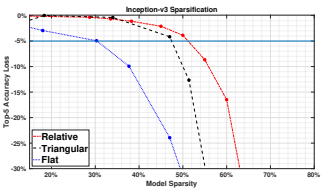
Relative Method In particular, it uses the δ^{th} percentile of distribution's weight in layer l denoted by δ_l. Thus each element of the vector $T(l) = \tau_l$ is defined as:

$$\tau_l = (max(\omega_l) - min(\omega_l)) \times \delta_l \tag{3}$$

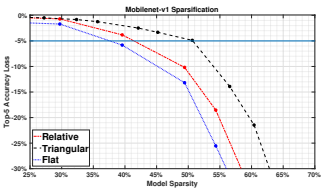
where $0 \leq \delta_l \leq 1$ defines the desired percentile of the δ_l of zero weights in each layers.

3 Experimental Evaluation and Comparison

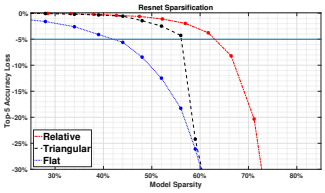
We evaluate our sparsification methods using TensorFlow v1.4 with CUDA runtime and driver v8. The evaluation of Top-5 accuracy is done on an NVIDIA's GeForce GTX 1060 with a host running Ubuntu 14.04, kernel v3.19 using ImageNet [13]. Figures 2a, 2b, 2c, 2d, and 2e show the inference accuracy as a function of the introduced sparsity by each method. They reflect that significant sparsity



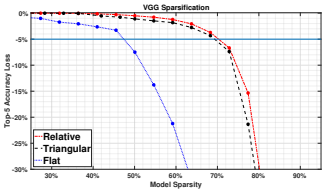
(a) Inception-v3



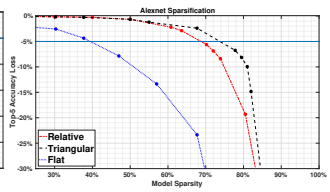
(b) MobileNets-v1



(c) ResNet-v2



(d) VGG-16



(e) Alexnet

Layers	#Params	%Sparsified
Conv1	23 K	10%
Conv2	307K	35%
Conv3	663K	35%
Conv4	1.3M	35%
Conv5	884K	35%
FC1	26M	85%
FC2	16M	85%
FC3	4M	73%
Total	**50M**	**81.1%**

(f) Alexnet Fine-tuning

Figure 2: Sparsity-Accuracy Trade-off for Our Three Proposed Sparsification Methods

can be obtained with a small reduction in inference accuracy. With less than 5% reduction in accuracy, we gain 51% sparsity (2.04× compression factor), 50% (2×), 62% (2.63×), 70% (3.33×), and 73% (3.7×) for the models. This validates our approach.

Further, the figures reflect that the relative method outperforms the other two methods for the Inception-v3 [25], VGG [20] and ResNet [9], but the triangular one outperforms the other two for MobileNet-v1 [11] and Alexnet [14]. This is likely due to the structure of the models. MobileNet-v1 and Alexnet have a gradual linear increase in the size of the convolution layers, making the triangular method more effective. In contrast, the other models have no such increase, making the relative method more effective. As a case-in-point, ResNet has 152 `conv` layers with variable sizes, which makes the triangular method less effective, as seen by the drop in accuracy in Figure 2c.

An interesting observation is that for Alexnet, introducing the first 50% sparsity suffers little drop in accuracy. This value is 35%, 30%, 41%, and 42% for the other models and it shows there exists significant redundancy within CNNs. Han et al. [6] observe the same with their Caffe implementation of Alexnet. The work was mainly focused on iteratively pruning and retraining CNNs to compensate the loss of accuracy. The authors' method with no retraining is not specified and it is unclear if it applies to other CNNs or not. However, the authors report a gain of around 80% sparsity by pruning (i.e., without retraining) Alexnet with L2 regularization. Our evaluation validates their result across other models using the proposed on-the-fly methods.

Fine-tuning. We explore what can be achieved by some fine-tuning of our methods, still with no retraining, in order to gain more sparsity. We do so to determine the effectiveness of our on-the-fly methods, since the fine-tuning is not likely feasible in out context. We focus on the relative method and start with a baseline sparsity. We then vary the degree of sparsity of each layer in turn around the base sparsity, attempting to maintain a no more than 5% drop in inference accuracy. The results for only AlexNet (due to space limitations) are shown in the table in Figure 2f. The baseline sparsity is selected as 70%. It is possible for some layers, particularly larger ones, to have higher sparsity, while smaller/earlier layers are more sensitive to sparsification and must have lower sparsity. Nonetheless, there is a gain of 8% in overall model sparsity. This value is 4%, 3%, 2% , and 5% for Inception-v3, Mobilenet-v1, ResNet, and VGG, respectively. Since this gain comes at the expense of an exploration of different sparsity ratios for the layers and thus more computations, it is not feasible in the contexts we explore. However, the gain is not significant to render our on-the-fly methods inefficient on their own without further tuning.

4 Concluding Remarks

In this paper, we proposed three model-independent methods to explore sparsification of CNNs without retraining. We experimentally evaluated these methods and showed that they can result in up to 73% sparsity with less than 5% drop in inference accuracy. However, there is no single method that works best for all models. Further, our evaluation showed that it is possible to fine-tune the methods to further gain sparsity with no significant drop in inference accuracy. However, such tuning of the methods cannot be employed on-the-fly. There are two key directions for future work. The first is to explore heuristics for selecting a sparsification method based on the CNN model and possibly fine tune the parameters of the methods using a predictive modeling. The second is to realize the benefit of the sparsity in the model's implementation on the NNlib library, which offloads neural networks operations from TensorFlow to Qualcomm's Hexagon-DSP.

References

[1] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. Fixed point optimization of deep convolutional neural networks for object recognition. In *2015 IEEE Int. Conf. Acoust. Speech Signal Process.*, pages 1131–1135. IEEE, apr 2015.

[2] Amir H. Ashouri, William Killian, John Cavazos, Gianluca Palermo, and Cristina Silvano. A survey on compiler autotuning using machine learning. *ACM Comput. Surv.*, 51(5):96:1–96:42, September 2018.

[3] Yann Le Cun, John S Denker, and Sara a Solla. Optimal Brain Damage. *Adv. Neural Inf. Process. Syst.*, 2(1):598–605, 1990.

[4] Xin Dong, Shangyu Chen, and Sinno Jialin Pan. Learning to Prune Deep Neural Networks via Layer-wise Optimal Brain Surgeon. *NIPS*, pages 4860–4874, may 2017.

[5] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A. Horowitz, and William J. Dally. EIE: Efficient Inference Engine on Compressed Deep Neural Network. In *Proc. - 2016 43rd Int. Symp. Comput. Archit. ISCA 2016*, pages 243–254. IEEE, jun 2016.

[6] Song Han, Huizi Mao, and William J. Dally. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *arXiv Prepr. arXiv1510.00149*, oct 2015.

[7] Tianyi David Han and Tarek S Abdelrahman. Automatic tuning of local memory use on gpgpus. *arXiv preprint arXiv:1412.6986*, 2014.

[8] Babak Hassibi and David Stork. Second order derivaties for network prunning: Optimal brain surgeon. In *Nips*, pages 164–171, 1993.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *2016 IEEE Conf. Comput. Vis. Pattern Recognit.*, pages 770–778, 2016.

[10] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[11] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *ArXiv*, 587:9, apr 2017.

[12] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size. In *arXiv*, feb 2017.

[13] Jia Deng, Wei Dong, R Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conf. Comput. Vis. Pattern Recognit.*, pages 248–255, 2009.

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. *Adv. Neural Inf. Process. Syst.*, pages 1–9, 2012.

[15] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[16] Huizi Mao, Song Han, Jeff Pool, Wenshuo Li, Xingyu Liu, Yu Wang, and William J. Dally. Exploring the Granularity of Sparsity in Convolutional Neural Networks. In *IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. Work.*, volume 2017-July, pages 1927–1934, may 2017.

[17] Huizi Mao, Song Han, Jeff Pool, Wenshuo Li, Xingyu Liu, Yu Wang, and William J. Dally. Exploring the Regularity of Sparse Structure in Convolutional Neural Networks. *arXiv Prepr. arXiv1705.08922*, may 2017.

[18] Manu Mathew, Kumar Desappan, Pramod Kumar Swami, and Soyeb Nagori. Sparse, Quantized, Full Frame CNN for Low Power Embedded Devices. In *IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. Work.*, volume 2017-July, pages 328–336, 2017.

[19] Vlad Niculae and Mathieu Blondel. A Regularized Framework for Sparse and Structured Neural Attention. *papers.nips.cc*, 2017.

[20] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *Int. Conf. Learn. Represent.*, pages 1–14, sep 2015.

[21] Yi Sun, Xiaogang Wang, and Xiaoou Tang. Sparsifying Neural Network Connections for Face Recognition. In *2016 IEEE Conf. Comput. Vis. Pattern Recognit.*, pages 4856–4864. IEEE, jun 2016.

[22] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S. Emer. Efficient Processing of Deep Neural Networks: A Tutorial and Survey. *Proc. IEEE*, 105(12):2295–2329, dec 2017.

[23] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, volume 4, page 12, 2017.

[24] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, volume 07-12-June, pages 1–9. IEEE, jun 2015.

[25] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the Inception Architecture for Computer Vision. *arXiv*, dec 2015.

[26] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning Structured Sparsity in Deep Neural Networks. *Nature*, 521(12):61–4, 2016.