JOINT OPTIMIZATION OF QUANTIZATION AND STRUCTURED SPARSITY FOR COMPRESSED DEEP NEURAL NETWORKS

Gaurav Srivastava, Deepak Kadetotad, Shihui Yin, Visar Berisha, Chaitali Chakrabarti, Jae-sun Seo

School of Electrical, Computer and Energy Engineering, Arizona State University, Tempe, AZ, USA

ABSTRACT

The usage of Deep Neural Networks (DNN) on resource-constrained edge devices has been limited due to their high computation and large memory requirement. In this work, we propose an algorithm to compress DNNs by jointly optimizing structured sparsity and quantization constraints in a single DNN training framework. The proposed algorithm has been extensively validated on high/low capacity DNNs and wide/deep sparse DNNs. Further, we perform Pareto-optimal analysis to extract optimal DNN models from a large set of trained DNN models. The optimal structurally-compressed DNN model achieves ∼50X weight memory reduction without test accuracy degradation, compared to floating-point uncompressed DNN.

1. INTRODUCTION

In recent years, DNNs have unprecedentedly improved accuracy in practical recognition and classification tasks, some even surpassing human-level accuracy [1, 2]. However, to achieve incremental accuracy improvement, state-of-the-art DNN algorithms tend to present very deep/large models (e.g., 1,000-layer networks [1]), which poses significant challenges for hardware implementations in terms of computation, memory, and communication. This is especially true for edge devices that exhibit severe constraints in area and power/energy.

This necessitates techniques to compress the DNN and lower energy consumption. A number of prior works investigated methods to lower the precision of activations/weights [3, 4] and apply pruning and compression [5] for DNNs, while maintaining high classification accuracy. Low-precision techniques quantize the DNN weights and activations. Quantizing DNNs to a very low precision (e.g. 1-2 bit) has been achieved by involving specific low-precision constraints during training [3, 4]. Large compression of DNNs was shown in [5] through pruning neurons and weights. However, the resulting scattered sparsity does not necessarily result in hardware acceleration [6, 7], and also increases the storage overhead for encoding sparsity [8]. Structured Sparsity Learning (SSL) [6] has demonstrated row-/column-/layer-wise structured sparsity based on group Lasso regularization, demonstrating enhanced acceleration. Coarse Grain Sparsity (CGS) was presented in [8], where static sparsity is applied on randomly selected blocks of weights throughout training, leading to large memory reduction with minimal index overhead. In [9], CGS and low-precision techniques were integrated during DNN training, but only a single DNN model was analyzed.

More recent works [10, 11] investigated another dimension – DNN model size – for precision-lowering [10] and pruning [11]. The authors of [10] reported that, for iso-accuracy, wider networks can lower the precision of activations/weights much more than shallower networks, so that the total computation cost actually becomes less. In [11], it has been shown that large-sparse models outperform small-dense models in test accuracy for the same memory footprint.

While these prior works separately investigated low-precision, structured compression, and/or DNN model size, there has been little work that systematically applied and optimized all of these three techniques in a single framework. For example, CGS [8] employed block-wise structured sparsity, but only quantized the weights and activations after training was complete, resulting in limited precision reduction (5-6 bit). To simultaneously achieve very low precision (1-3 bit) and structured sparsity in DNNs without hampering accuracy, all three dimensions need to be explored holistically and applied throughout the training process. As shown in [10, 11], for the same accuracy, smaller networks do not necessarily result in smaller footprint or less computation when collectively analyzed with low precision [10] or sparsity [11].

In this work, we jointly optimize both structured sparsity and quantization techniques in a single training framework for various DNN model sizes. The objective is to find the optimal DNN algorithm that can be implemented in efficient hardware with minimal area/energy for resource-constrained edge devices. Among the large number of trained DNNs (107 DNNs) that we explored, we found 11 optimal designs using the Pareto-optimal approach, and further analysis on the DNN memory breakdown (e.g. weight versus activation memory, convolution versus fully-connected layers) has been performed on those optimal DNNs. One of the key results shows that structurally-compressed DNN with 2-bit weight precision achieves overall 50X weight memory reduction, compared to floating-point uncompressed DNN, while obtaining >92% test accuracy for CIFAR-10 dataset. Our source code can be found at `https://gitlab.com/srivastavag/CGS_quantization`.

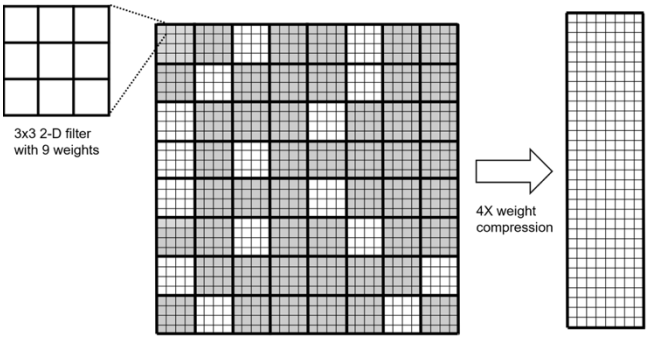


Fig. 1: Illustration of CGS mask for a 32x32 weight matrix with CGS size 4x4 and CGS ratio 4X. The gray and white blocks represent zero and non-zero blocks, respectively.

To put our proposed work into the perspective of DNN hardware implementation, let us consider mapping representative DNNs onto FPGA-based edge devices. For example, performing inference on a ResNet-50 [1] DNN requires storage of ∼26M weight parameters and ∼16M activations, and a 32-bit floating-point ResNet-50 would need 168MB of memory. If we consider edge devices based on Intel Arria 10 FPGA, the maximum FPGA on-chip memory storage is 8.4MB [12], so the baseline ResNet-50 would not fit in the FPGA. This necessitates a large amount of DRAM access, which can be a dominant portion of the overall energy and latency [13][1]. The overall 50X model reduction achieved in this work can enable on-chip storage of all weights/activations on edge devices and the structured sparsity based compression will enable sufficient hardware acceleration [6, 7].

2. JOINT-OPTIMIZATION OF QUANTIZATION AND STRUCTURED SPARSITY FOR DNN

2.1. Proposed DNN Training Framework

The proposed DNN training algorithm uses the BNN training scheme [4] with additional constraints on CGS [8] based structured sparsity. Prior to training, blocks of weights were randomly dropped as illustrated in Fig. 1 (CGS mask), depending on CGS size and CGS ratio. CGS size is the shape of a CGS block and CGS ratio is the compression ratio of non-zero blocks to all blocks. The weights of the dropped blocks statically remain zero throughout the entire training process and classification. With regards to CGS mask for convolution layers, the smallest rectangle in Fig. 1 is the mask applied to all weights in a 2D filter (e.g. 3×3). The smallest rectangle in the CGS mask for fully-connected layers represents a single weight. Quantization constraints are applied when we train the non-zero weight blocks (white color in Fig. 1).

The sparse weight matrix/tensor, generated after applying CGS, is trained using backpropagation. Quantized weights are generated from high-precision weights using a quantization function for the target precision. Once the activations are computed, they are quantized to the specified activation precision. The quantized weights/activations are used for the forward phase. During the backward phase, gradients of the cost function with respect to activations/weights are computed and backpropagated. Straight-through estimator [14] is used to estimate the gradient with respect to quantized activations. During the weight update phase, the high-precision weights are updated only for non-zero blocks of weights, using Eq. 1.

$$(W_{ij})_{k+1} = (W_{ij})_k + \{(\Delta W_{ij})_k + m \times (\Delta W_{ij})_{k-1}\} \times lr \times C_{ij}, \quad (1)$$

where $(W_{ij})_k$ is weight at k^{th} iteration, m is momentum, lr is learning rate, and C_{ij} is the CGS connection coefficient between two consecutive DNN layers. $C_{ij} = 0$ for weights corresponding to the zero blocks, and $C_{ij} = 1$ for weights corresponding to non-zero blocks.

2.2. Experiment Setup

The proposed algorithm has been evaluated on CIFAR-10 dataset for image classification tasks. We have used deep learning framework Theano [15] and toolbox Lasagne [16] for DNN training and classification. We performed experiments for different weight/activation precision values, including floating-point (W-FP:A-FP), 8-bit (W8b:A8b), 4-bit (W4b:A4b), 2-bit (W2b:A2b), and 1-bit (W1b:A1b), as well as different CGS ratios of 32X, 16X, 8X, 4X, 2X, and 1X, where 1X is an uncompressed DNN. Furthermore, we compare the performance of wide-sparse, thin-dense, deep-sparse, and shallow-dense DNNs. Wide-sparse DNNs have larger number of neurons in a layer and higher CGS ratio compared to thin-dense DNNs, so that the total weight memory is comparable. Similarly, deep-sparse DNNs have larger number of layers and higher CGS ratio compared to shallow-dense DNNs. The weight memory reported in all of our results includes the index memory needed for CGS non-zero blocks.

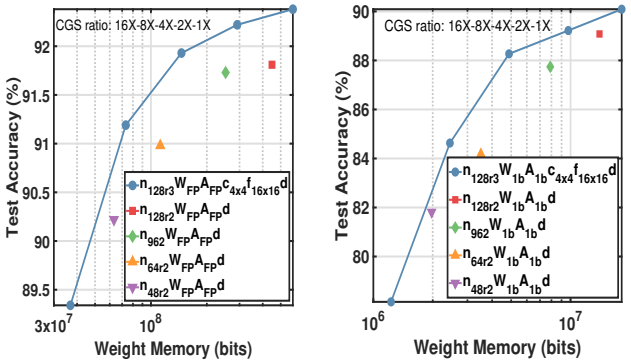
Experiments were performed on DNN models of three different depths, e.g. n_{128r3}, n_{128r2} and n_{128r1}. Here, n_{128r2} represents a DNN that has 128 feature maps in first convolution layer and has 2 repeated convolution layers with same width before doubling the number of feature maps, i.e. C128-C128-P2-C256-C256-P2-C512-C512-P2-F1024-F1024-F10. n_{128r3} and n_{128r1} has 3 and 1 repeated convolution layers, respectively. In addition, DNNs with varying widths (different number of feature maps) for each of the three depths (r1, r2, r3), such as n_{32r3} and n_{64r2}, have been trained and analyzed. Dropout ratio of 20% is employed on convolution layers and 50% on fully-connected layers.

3. EXPERIMENTAL RESULTS

3.1. Comparison of Wide-Sparse / Thin-Dense DNNs

We first compare wide-sparse and thin-dense DNNs. Wide-sparse floating-point high-capacity (r3 depth) DNN resulted

[1] ∼40% of total latency was reportedly consumed by DRAM access for end-to-end ResNet-50 implementation on Arria 10 FPGA in [13].



(a) (b)

Fig. 2: Test accuracy versus weight memory comparison between deep-sparse (n_{128r3}) and shallow-dense (other data points) DNNs are shown for (a) floating-point (W-FP:A-FP) and (b) 1-bit weight/activation precision (W1b:A1b).

in ∼2X memory reduction at 92% iso-accuracy, compared to a thin-dense floating-point DNN. However, wide-sparse DNN gives minimal or no memory savings for low precision schemes (1-bit, 2-bit, 4-bit, 8-bit). Moreover, for mid-capacity (r2 depth) DNNs, wide-sparse DNNs show inferior accuracy compared to thin-dense DNNs.

3.2. Comparison of Deep-Sparse / Shallow-Dense DNNs

On the other hand, we observed accuracy/memory benefits when using deep-sparse DNNs compared to shallow-dense DNNs. First, we investigated mid-capacity DNNs, comparing deep-sparse (r2 depth) versus shallow-dense (r1 depth) DNNs. Test accuracy of floating-point deep-sparse r2 DNN (W-FP:A-FP) is ∼7% higher than that of shallow-dense r1 DNN for 40Mb weight memory DNN. Similar observations were made for mid-capacity low-precision DNNs (<8-bit). For example, 1-bit deep-sparse DNN (1.5Mb weight memory) demonstrated ∼13% higher test accuracy than that of shallow-dense DNN.

We observed similar trends for high-capacity DNNs, when we compared deep-sparse (r3 depth) and shallow-dense (r2 depth) DNNs. As seen in Fig. 2a, high-capacity deep-sparse floating-point DNN (W-FP:A-FP) demonstrates 0.5% higher accuracy across all sparsity levels. For low precision schemes (<8-bit) on high-capacity r3 DNNs, we observed 2-3X memory savings at iso-accuracy. 1-bit DNN (Fig. 2b) shows ∼3X weight memory savings at ∼88% accuracy. Overall, we find that memory saving diminishes with lower precision DNNs. In addition, with more zero weights and lower bit precision, arithmetic operations/computations will also reduce and result in lower power consumption.

3.3. Comparison of DNNs with Quantization and CGS

To obtain the optimal quantization and structured sparsity values, a number of DNN models are analyzed together for iso-accuracy. Fig. 3 presents test accuracy versus weight memory trade-off on high-capacity (n_{128r3}) DNN models for different CGS ratio and precision configurations. Floating-point uncompressed model is the baseline DNN model on which low-precision and CGS constraints were applied. Applying only CGS constraints on floating-point uncompressed DNN achieves 2X compression with <0.16% accuracy degradation. If only quantization is used, 16X compression was achieved with 0.80% accuracy improvement for 2-bit DNN. By jointly optimizing quantization and low-precision constraints, memory saving of ∼50X was achieved for the 2-bit precision and 4X CGS ratio scheme without any accuracy degradation.

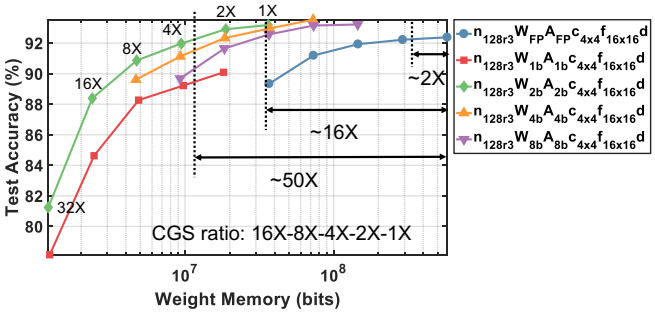


Fig. 3: Test accuracy versus weight memory comparison for different precision and structured sparsity configurations on high-capacity n_{128r3} DNN.

Similar analysis was performed on mid-capacity (n_{128r2}) DNN with different CGS ratio and precision settings. Compared to the uncompressed DNN, ∼2X weight memory compression is achieved by using only CGS compression, without degrading test accuracy. Using only low-precision quantization, compared to floating-point DNN, ∼16X weight memory savings is achieved with accuracy gain of 0.80% for 2-bit precision DNN, likely due to regularization [4]. Overall 32X weight savings is achieved without any test accuracy degradation by simultaneously optimizing quantization and CGS.

3.4. DNN memory analysis

For DNN accuracy and memory optimization, we employ the Pareto-optimal approach to extract the optimal DNN designs and then perform memory analysis on the optimal designs. We trained a large number (107) of DNN models with varying depth and width, to comprehensively analyze the trade-off between test accuracy and memory utilization. Fig. 4a shows the Pareto-optimal plot for all 107 trained DNNs. The Pareto-front, shown by the red line, maximizes test accuracy and minimizes weight memory. By choosing all the designs close to the Pareto-front, we obtained 11 optimal DNN designs. It is noteworthy that the optimal designs do not include any low-capacity (r1 depth) DNNs. Total DNN memory is the sum of activation memory and weight memory. While only weight memory was considered for extracting the Pareto-optimal designs, we analyzed the contribution of activation memory for the 11 optimal designs. The analysis was performed using batch size of 1, targeting inference on edge devices.

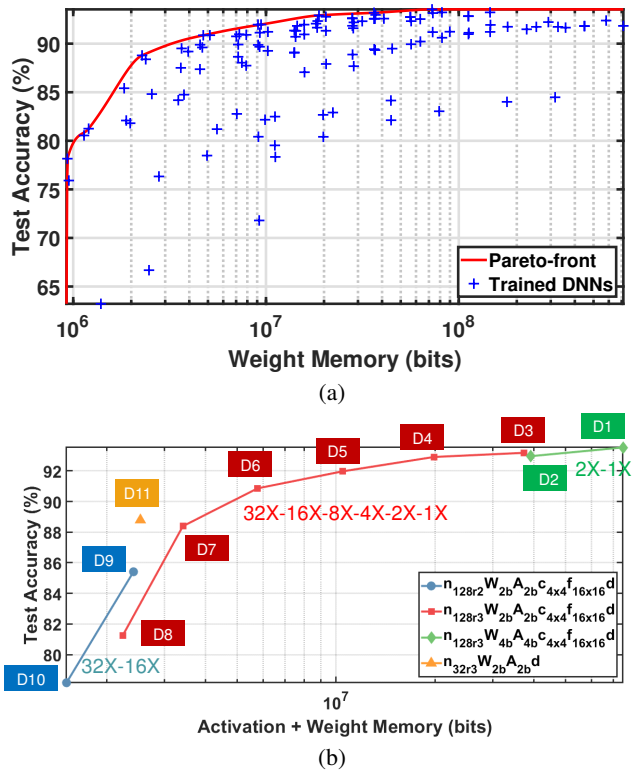


Fig. 4: (a) Pareto-front of all compressed and uncompressed DNNs. (b) Test accuracy versus activation and weight memory for 11 optimal designs on the Pareto front.

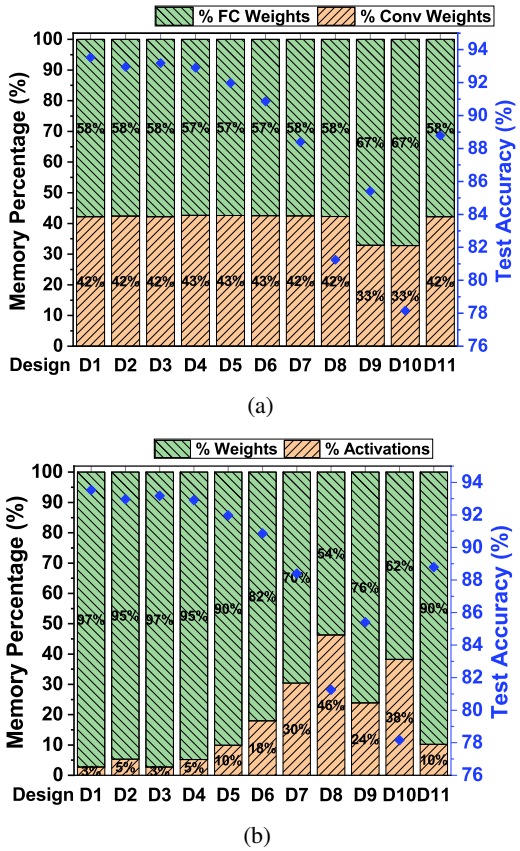


Fig. 5: (a) Distribution of convolution versus fully-connected weight memory for 11 optimal designs. (b) Distribution of activation versus weight memory for 11 optimal designs.

Fig. 4b shows test accuracy as a function of sum of activation memory and weight memory for inference batch size of 1. After adding activation memory to weight memory, there is only a small change in the memory footprint of the optimal designs corresponding to the low sparsity networks (CGS ratio 1X-4X). However, if the network has high sparsity (CGS ratio 8X-32X) the effect of activation memory on overall DNN memory cannot be ignored.

In DNNs, typically convolution layers contribute for heavy computations and fully-connected layers dominate the DNN memory footprint. Fig. 5a shows the weigth distribution of convolution versus fully-connected layers for the 11 optimal designs. The ratio of convolution weights versus fully-connected weights is slightly lower for high-capacity (r3 depth) DNN models. When the optimal design is mid-capacity (r2 depth) DNN, the contribution from fully-connected weights dominate the weight memory distribution. However, weight distribution of convolution versus fully-connected layers can depend on the DNN architecture. Fig. 5b shows the distribution of activation memory and weight memory for the optimal designs for inference batch size of 1. The ratio of activation memory to that overall memory is insignificant for low sparsity DNNs (CGS ratio 1X-4X). However, at high sparsity (CGS ratio 8X-16X), the activation memory becomes a significant component of the overall DNN memory as weights get pruned largely, while all the activations remain non-sparse.

4. CONCLUSION

High computation and large memory requirement of DNNs prohibit their deployment on resource-constrained edge devices. To enhance compression and hardware acceleration, this work jointly optimizes structured sparsity and quantization in a single DNN training framework. By investigating varying-depth/width DNNs with low precision and structured sparsity, 50X weight memory reduction is achieved without accuracy degradation, compared to floating-point uncompressed DNN. Our experiments demonstrated that deep-sparse DNN outperform shallow-dense DNN with comparable weight memory. However, the weight memory savings diminish as the precision is reduced and structured sparsity is increased. Future work includes applying the algorithm on other DNN architectures and large-scale ImageNet dataset.

5. ACKNOWLEDGEMENT

This work is in part supported by NSF grant 1652866, Samsung, C-BRIC, one of six centers in JUMP, a SRC program sponsored by DARPA, and ONR grant N00014-17-1-2826.

6. REFERENCES

[1] K. He, X. Zhang, S.g Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[2] W. Xiong, L. Wu, F. Alleva, J. Droppo, X. Huang, and A. Stolcke, "The Microsoft 2017 conversational speech recognition system," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018.

[3] M. Courbariaux, Y. Bengio, and J.-P. David, "BinaryConnect: Training deep neural networks with binary weights during propagations," in *Advances in Neural Information Processing Systems (NIPS)*, 2015.

[4] I. Hubara, M.u Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Advances in Neural Information Processing Systems (NIPS)*, 2016.

[5] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," in *International Conference on Learning Representations (ICLR)*, 2016.

[6] 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 (NIPS)*, 2016.

[7] J. Yu, A. Lukefahr, D. Palframan, G. Dasika, R. Das, and S. Mahlke, "Scalpel: Customizing DNN pruning to the underlying hardware parallelism," in *IEEE/ACM International Symposium on Computer Architecture (ISCA)*, 2017.

[8] D. Kadetotad, S. Arunachalam, C. Chakrabarti, and J. Seo, "Efficient memory compression in deep neural networks using coarse-grain sparsification for speech applications," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2016.

[9] S. Yin, G. Srivastava, S. K. Venkataramanaiah, C. Chakrabarti, V. Berisha, and J. Seo, "Minimizing area and energy of deep learning hardware design using collective low precision and structured compression," in *IEEE Asilomar Conference on Signals, Systems, and Computers*, 2017.

[10] A. Mishra, E. Nurvitadhi, J. J. Cook, and D. Marr, "WRPN: Wide reduced-precision networks," in *International Conference on Learning Representations (ICLR)*, 2018.

[11] M. Zhu and S. Gupta, "To prune, or not to prune: exploring the efficacy of pruning for model compression," *arXiv preprint arXiv:1710.01878*, 2017.

[12] Intel Corporation, "Intel Arria 10 device overview," `https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/arria-10/a10_overview.pdf`.

[13] Y. Ma, Y. Cao, S. Vrudhula, and J. Seo, "An automatic RTL compiler for high-throughput FPGA implementation of diverse deep convolutional neural networks," in *International Conference on Field Programmable Logic and Applications (FPL)*, 2017.

[14] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *arXiv preprint arXiv:1308.3432*, 2013.

[15] Theano Development Team, "Theano: A Python framework for fast computation of mathematical expressions," *arXiv preprint arXiv:1605.02688*, May 2016.

[16] S. Dieleman, J. Schlter, C. Raffel, E. Olson, S. K. Snderby, D. Nouri, et al., "Lasagne: First release," Aug. 2015.