Scalable Compression of Deep Neural Networks

Xing Wang
Simon Fraser University, BC, Canada
AltumView Systems Inc., BC, Canada
xingw@sfu.ca

Jie Liang
Simon Fraser University, BC, Canada
AltumView Systems Inc., BC, Canada
jiel@sfu.ca

ABSTRACT

Deep neural networks generally involve some layers with millions of parameters, making them difficult to be deployed and updated on devices with limited resources such as mobile phones and other smart embedded systems. In this paper, we propose a scalable representation of the network parameters, so that different applications can select the most suitable bit rate of the network based on their own storage constraints. Moreover, when a device needs to upgrade to a high-rate network, the existing low-rate network can be reused, and only some incremental data are needed to be downloaded. We first hierarchically quantize the weights of a pre-trained deep neural network to enforce weight sharing. Next, we adaptively select the bits assigned to each layer given the total bit budget. After that, we retrain the network to fine-tune the quantized centroids. Experimental results show that our method can achieve scalable compression with graceful degradation in the performance.

CCS Concepts

•**Computing methodologies** → **Neural networks;** *Discrete space search;* •**Information systems** → Clustering and classification;

Keywords

Deep neural network, scalable compression

1. INTRODUCTION

Deep neural networks (DNN) or deep learning have evolved into the state-of-the-art technique for many artificial intelligence tasks including computer vision [9], [14]. In this paper, we focus on the convolutional neural network (CNN), which was originally developed in 1998 by LeCun et al. with less than 1M parameters (weights) to classify handwritten digits [10]. In 2012, CNN was used as a key component in [9] to achieve the breakthrough in ImageNet Large Scale Visual Recognition Challenge 2012 (ILSVRC2012), and the proposed AlexNet has 60M parameters and needs 240MB of storage space. In 2014, Simonyan et al. further improved the accuracy by 10% [14], and the VGG-16 model they developed has 138M parameters.

Although the performance of DNN is very promising, its application in low-end devices such as mobile phones faces some challenges. For example, many devices have limited storage spaces. Therefore storing millions of DNN parameters on these devices could be a problem. If the DNN network needs to be updated, usually via wireless channels, downloading the large amount of network parameters will cause excessive delay. Moreover, running large-scale DNNs with floating-point parameters could consume too much energy and slow down the algorithm. Therefore, efficient compression of the DNN parameters without sacrificing too much the performance becomes an important topic.

There have been some recent works on the compression of neural networks. Vanhoucke et al. [11] proposed a fixed-point implementation with 8-bit integer (vs 32-bit floating-point) activations. Denton et al. [3] exploited the linear structure of the neural network by finding an appropriate low-rank approximation of the parameters and keeping the accuracy within 1% of the original model. Kim et al. [7] applied tensor decomposition to the network parameters and proposed an one-shot whole network compression scheme that can achieve significant reductions in model size, runtime and energy consumption.

Many works focus on binning the network parameters into buckets, and only the values in the bucket need to be stored. HashedNets [2] is a recent technique to reduce model size by using a hash function to randomly group connection weights, so that all connections within the same hash bucket share a single parameter value. Gong et al. [4] compressed deep convnets using vector quantization, which resulted in 1% accuracy loss. Both methods studied the fully-connected (FC) layer in the CNN, but ignored the convolutional (CONV) layers. Recently, Han et al. [5] introduced a deep neural network compression pipeline by combining pruning, quantization and Huffman encoding, which can reduce the storage requirement of neural network by 35 × or 49 × without affecting their accuracy.

In this paper, motivated by the successful applications of scalable coding in various image and video coding standards such as JPEG 2000, H.264, and H.265/HEVC [16, 13, 15], we propose a scalable compression framework for DNNs, which has not been addressed before. Our goal is to represent the DNN parameters in a scalable fashion such that we can easily truncate the representation of the network according

MM '16, October 15-19, 2016, Amsterdam, Netherlands

DOI: http://dx.doi.org/10.1145/2964284.2967273

to the storage constraint and still get near-optimal performance at each rate. Moreover, if the network needs to be upgraded with higher rate and better performance, the existing low-rate network can be reused, and only some incremental data are needed. This is better than recompressing and re-transmitting the network as in [5, 7].

To achieve this goal, we propose a three-stage pipeline. First, a hierarchical representation of weights in DNNs is developed. Second, we propose a backward greedy search algorithm to adaptively select the bits assigned to each layer given the total bit budget. Finally, we fine-tune the compressed model.

The rest of the paper is organized as follows. Sec. 2 is devoted to hierarchical quantization of the DNN parameters. In Sec. 3 we formulate the bit allocation as an optimization problem and propose a backward search solution. A fine-tuning method is presented in Sec. 4. Experimental results on MNIST, CIFAR-10 and ImageNet datasets are reported in Sec. 5, followed by conclusions in Sec. 6.

2. HIERARCHICAL QUANTIZATION

The K-means clustering-based quantization is a popular technique in the compression of DNN [4], [5]. Therefore, in this paper, we also choose K-means clustering with linear initialization [5] to compress the weights in DNN. However, the framework developed in this paper is quite general and can also be applied to other quantization techniques, e.g., the fixed-point quantization in [11] and other similar tasks besides classification, e.g., regression problems.

In [5], the authors quantize the weights to enforce weight sharing with K-means clustering, e.g., they assign 8 bits (256 shared weights) to each CONV layer and 5-bits (32 shared weights) to each FC layer. However, every time a CONV layer is assigned a different bit, the K-means clustering has to be performed again, rendering scalable compression infeasible. On the other hand, some DNN layers have a large number of weights, e.g., the number of weights in the fc6 layer of AlexNet is 38M. Therefore the K-means clustering can be quite slow, even with the help of GPU.

To address this problem, we adopt the scalable coding concept in image/video coding [16, 13, 15], and represent the weights hierarchically, i.e., each weight is represented by a base-layer component and several enhancement-layer components; hence, we only need to perform the quantization step once during the entire scalable compression process, which also benefits the adaptive bit allocation in Sec. 3. Note that there are two different kinds of layers in this paper: the network layers in DNN, and the hierarchical quantization layers in the scalable representation of the weights.

Suppose we want to allocate n bits to each weight in a pre-trained DNN layer. We first perform K-means clustering of all weights with $K = 2$ (1-bit quantization), and record the corresponding cluster indices and centroids. We also record the corresponding quantization error. This yields the 1-bit base-layer approximations of all weights. Next, we perform another K-means clustering with $K = 2$ on all quantization errors, and record the corresponding cluster indices, centroids, and quantization errors. The gives us the 1-bit first-enhancement-layer representations of all weights. By repeating this procedure, we can obtain a n-layer hierarchical representation of a weight, i.e.,

$$w \approx b_1 + e_1 + ... + e_{n-1}, \tag{1}$$

where w is a uncompressed weight, b_1 and e_i are the centroid of the base layer and the i-th enhancement layer respectively.

This hierarchical quantization only needs to be performed once, which facilitates future network updating, as we only need to add or delete certain quantization layers to meet the new bit rate constraint. For the tradition K-means clustering used in [4] and [5], we have to perform K-means clustering every time a new bit budget is required.

After the hierarchical quantization, we can build a codebook that stores the centroid and cluster index information of all quantization layers. For a network layer of DNN with N weights, there are $2n$ centroids, and the number of cluster indices is Nn. If each uncompressed weight or centroid is represented by b bits (b=32 for single-precision floating-point number), the compression rate of the n-bit hierarchical quantization scheme is

$$r = \frac{Nb}{Nn + 2nb}. \tag{2}$$

In contrast, in the conventional K-means method [5], given the same n-bit quantization, the compression rate is $r = Nb/(Nn + 2^n b)$.

Note that the storage cost is dominated by Nn, compared to $2nb$ or $2^n b$, because the number of connections N in a DNN is usually very large.

3. ADAPTIVE BIT ALLOCATION

In DNN, the redundancies in different network layers are different [5, 11]. Therefore it is necessary to design an optimal bit allocation algorithm, i.e., given a bit budget, how to allocate the bits to different network layers in order to get the best performance. In this paper, we formulate the following optimization problem.

$$\begin{aligned} &\arg \min_{\{\mathbf{n},\mathbf{C},\mathbf{G}\}} f(\mathbf{n}, \mathbf{C}, \mathbf{G}) \\ &\text{s.t.} \sum_{i=1}^{L} N_i n_i + 2n_i b \le \mu. \end{aligned} \tag{3}$$

where $\mathbf{n} = \begin{bmatrix} n_1 & ... & n_L \end{bmatrix}$ is a vector containing the bits allocated to L network layers, $\mathbf{C}$ is the centroid vector, $\mathbf{G}$ is the cluster-by-index matrix for the network layers, N_i is the number of weights in the i-th network layer, and μ is the bit budget. We use the cross entropy between the pdf of the predicted labels and true labels as the cost function $f(\cdot)$, which is frequently used in classification tasks.

It is hard to solve the combinatorial optimization in Eq. (3), since the number of bits assigned to each network layer n_i has to be integer and the number of entries in the cluster-by-index matrix $\mathbf{G}$ is $\sum_{i=1}^{L} N_i n_i$, even larger than the number of weights $\sum_{i=1}^{L} N_i$ in the pre-trained DNN. Therefore, we use a similar method to [5] to first approximate the original uncompressed weights with high-rate quantized weights. More specifically, we first use the hierarchical method in Sec. 2 to assign M bits to each CONV layer weight and P bits to each FC layer weight. This is used as the initialization step. The centroid vector $\mathbf{C}$ and the cluster-by-index matrix $\mathbf{G}$ are then determined and fixed. We use E to denote the number of bits to store this initial network.

Next, we adaptively allocate bits to network layers such

that $u < \mathrm{E}$. The problem in Eq. (3) is simplified to

$$\begin{aligned} &\arg\min_{\{\mathbf{n}\}} f(\mathbf{n}) \\ \text{s.t. } &B = \sum_{i=1}^{L} (N_i + 2b) n_i \leq \mu. \end{aligned} \tag{4}$$

For small-scale problems, the optimization above can be solved by exhaustive grid search, where configurations that violate the bit constraint are skipped, and the others are evaluated to find the best solution. The process can be accelerated by parallel computing, since different configurations are independent. However, for large-scale problems, exhaustive search becomes infeasible, as the number of configurations grows exponentially with the number of bits. For example, in AlexNet, there are 5 CONV layers and 3 FC layers. If 10 bits are assigned to each CONV layer and 5 bits are assigned to each FC layer, the total number of configurations would be $10^5 \times 5^3 = 12.5$M.

One way to speed up the process is to use random search [1], since the number of bits assigned to each network layer can be treated as a hyper-parameter for the DNN. Theoretical analysis in [1] shows that randomly selecting 60 configurations can ensure that the top 5% result can be achieved with a probability of 0.95. For the bit allocation problem here, we should randomly select a number of configurations that satisfy the bit constraint. In Sec. 5, random search is used as a baseline algorithm for comparison.

In this paper, we propose a backward greedy search algorithm to address the bit constraint explicitly and solve the problem in Eq. (4). We start from the initial high-rate quantized network as discussed above. Denote the bit allocation in the t-th iteration as $\mathbf{n}^t = [n_1^t, \ldots, n_L^t]$, whose corresponding total bit cost is B^t. To find $\mathbf{n}^{t+1}$ at iteration $t+1$, we follow the spirit of the gradient descent method by assigning one less bit to each network layer respectively, calculating the corresponding gradient of the total bit cost, and choosing the configuration that has the maximum gradient. In other words, let $\mathbf{n}^{t,j} = [n_1^t, \ldots, n_{j-1}^t, n_j^t - 1, n_{j+1}^t, \ldots, n_L^t]$, the bit allocation in the $(t+1)$-th iteration is obtained by

$$\begin{aligned} &\arg\max_{\mathbf{n}^{t,j}} \frac{f(\mathbf{n}^{t,j}) - f(\mathbf{n}^t)}{B^{t,j} - B^t} \\ \text{s.t. } &\mathbf{n}^{t,j} \subset \{\mathbf{n}^{t,1}, \mathbf{n}^{t,2}, ..., \mathbf{n}^{t,L}\}, \\ &B^{t,j} = \sum_{i=1}^{L} (N_i + 2b) n_i^{t,j}. \end{aligned} \tag{5}$$

The iteration terminates until the bit constraint is satisfied. The entire backward greedy search algorithm is summarized in Alg. 1. The intuition behind the gradient defined above is twofold. First, if two bit allocations have the same cost function value, the one with smaller total bit cost should be chosen. Second, if two bit allocations have the same total bit cost, we should choose the one with lower cost function value and use the maximum function in Eq. (5) due to $B^{t,j} < B^t$.

Algorithm 1 Backward Greedy Search Algorithm

Initialization: Quantize the network with M bits for each CONV layer and P bits for each FC layer. Let $t = 0$.
while $B^t > \mu$ **do**
  for each network layer $j \leq L$ **do**
    $n_j^{t,j} \leftarrow n_j^t - 1, n_p^{t,j} \leftarrow n_p^t$ for $p \neq j$.
    Update the weights of DNN based on the hierarchical framework in Sec. 2
    Test with the validation data and record $B^{t,j}$ and $f(\mathbf{n}^{t,j})$
  end for
  Select $\mathbf{n}^{t+1}$ based on Eq. (5)
  $t \leftarrow t + 1$
end while

4. FINE TUNING (FT)

It is shown in [5, 11] that fine-tuning (FT) of the centroids after the quantization of DNN can significantly improve the classification performance. In this paper, we also perform fine-tuning after the adaptive bit allocation to update the centroids based on Eq. (3) in [5].

The advantage of the proposed scalable compression of the DNN is that for each target bit rate, we can find a near-optimal bit allocation. If later on the DNN bit rate on a device needs to be updated, instead of re-transmitting a new set of the DNN parameters, we only need to transmit some incremental data, including the centroid vector $\mathbf{C}$ and cluster-by-index matrix $\mathbf{G}$. The required bit rate is thus much lower than replacing the entire network.

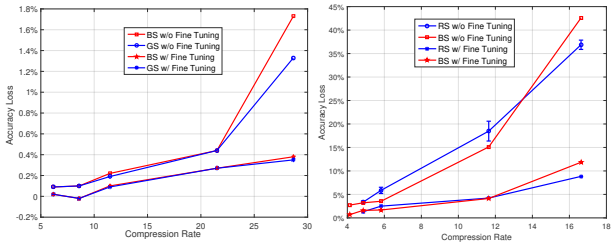
During the update, some additional bits caused by the fine-tuning are needed to update the centroids of the previous compressed model. However, according to the analysis in Sec. 2, the centroid update will cost $2b \sum_{i=1}^{L} n_i$ at most, while the minimal bits needed to update the cluster-by-index matrix are $\min\{N_1, N_2, ..., N_L\}$. The storage cost is dominated by the cluster indices instead of centroids; hence the overhead introduced by the fine-tuning is negligible. Take AlexNet as an example, if we use 10 bits to quantize CONV layers and 5 bits for FC layers, at most 0.52KB are needed to update these centroids, while we may use at least 5KB to update the cluster-by-index matrix every time a different bit budget is given.

5. EXPERIMENTAL RESULTS

We test the proposed scalable compression on 3 networks designed for the MNIST [10], CIFAR-10 [8] and ImageNet [12] datasets respectively. We implement the network training based on the CNN toolbox MatConvNet [17] with our own modifications. The training is done on a desktop with a NVIDIA TIAN X GPU with 12GB memory.

5.1 LeNet-5 for MNIST

We use the *cnn_mnist_experiment.m* function in MatConvNet to train LeNet-5 for MNIST dataset. There are 2 CONV layers and 2 FC layers. The pre-trained model can achieve 0.88% Top-1 error and needs a storage of 1720KB. We use 8 bits to hierarchically quantize each CONV layer and 5 bits for each FC layer. The initial quantized model can achieve 0.97% Top-1 error, and the corresponding storage cost is 279KB. In Fig. 1(a), we compare the proposed backward greedy search method (BS) with the exhaustive grid search method (GS). We also present the number of configurations tested on the validation set in Table 1 to compare the computational complexity. We can see that our proposed backward search algorithm can achieve comparable compression performance to the grid search with much smaller computational complexity. The only exception happens when the compression rate is extremely large,



(a) (b)

Figure 1: Top-1 accuracy loss of compressed DNNs under different bit allocation methods. (a) LeNet-5 and (b) CIFAR-10-quick.

LeNet-5 for MNIST				
Bit Budget (KB)	200	150	80	60
Compression Rate	8.60	11.47	21.50	28.67
BS Number	26	51	101	115
GS Number	960	640	320	79
CIFAR-10-quick for CIFAR-10				
Bit Budget (KB)	120	100	50	30
Compression Rate	4.85	5.82	11.64	19.40
BS Number	26	51	101	115
RS Number	120			

Table 1: Number of configurations tested on MNIST and CIFAR-10 validation set v.s. compression rate.

e.g., 28.67 in Fig. 1(a). However, after fine tuning, the performance is still very close to the original one.

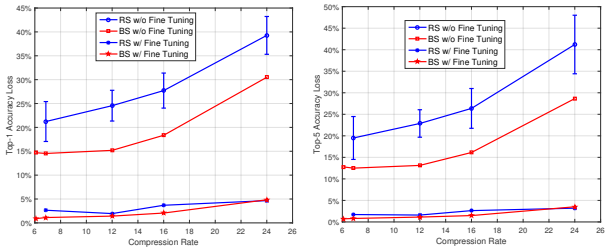
5.2 CIFAR-10-quick for CIFAR-10

We use the provided *cnn_cifar.m* in MatConvNet to train CIFAR-10-quick for CIFAR-10 dataset. There are 3 CONV layers and 2 FC layers in the network. The reference model can achieve 19.97% Top-1 error and needs a storage space of 582KB. We use 10 bits to quantize each CONV layer and 5 bits for each FC layer. The initial quantized model can achieve 22.70% Top-1 error and needs 141KB storage space. Since there are at most 25K configurations which takes too much time to evaluate, instead of using grid search as a comparison, we use the random search method (RS) [1]. In each trial, we randomly choose 120 configurations that satisfy the bit constraint from the configuration pool.

The compression performance is shown in Fig. 1(b) and the computational complexity is presented in Table 1. It can be seen that the proposed backward search algorithm can achieve similar or even better performance than random search with much smaller computational complexity, especially when the bit rate is close to that of the initial quantized network. The only exception happens when the compression rate is extremely large, e.g., 20 in Fig. 1(b). For the fine-tuning in the random search method, we fine-tune the result that achieves the median classification accuracy in the 10 trials.

5.3 AlexNet for ILSVRC12

We use the provided *cnn_imagenet.m* to train AlexNet for ILSVRC12. The reference model is slightly different from that of the original AlexNet in [9], where the order of pooling layer and norm layer are swapped. It contains 5 CONV layers and 3 FC layers. This reference model can achieve



(a) (b)

Figure 2: Accuracy loss of compressed AlexNet v.s. compression rate.

Bit Budget (MB)	35	20	15	10
Compression Rate	6.86	12	16	24
BS Number	25	81	89	126
RS Number	150			

Table 2: Number of configurations tested on ILSVRC12 validation set v.s. compression rate.

41.39% Top-1 error, 18.85% Top-5 error, and needs 240 MB to store. We use 10 bits to quantize each CONV layer and 5 bits for each FC layer. This initial quantized model can achieve 56.09% Top-1 error, 31.63% Top-5 error, and needs 39.5 MB to store. The number of configurations in each trial of RS is 150. The number of trials is 5 in order to get 0.95 confidence interval. The result that achieves the median classification accuracy in the 5 trials is fine-tuned.

The compression performance is shown in Fig. 2, and the computational complexity is shown in Table 2. We can see that with much smaller computational complexity, the proposed backward search can achieve better compression performance than random search. Moreover, the classification performance of proposed scalable compression framework drops little when the compression rate is within 10. The compression performance is comparable to state-of-the-art algorithms at fixed rate, e.g., AlexNet is compressed to 47.6 MB with more than 1% Top-1 accuracy loss in [3].

6. CONCLUSIONS AND FUTURE WORK

In this paper, we discuss the scalable compression of deep neural networks, and propose a three-stage pipeline: hierarchical quantization of weights, backward search for bit allocation, and fine-tuning. Its efficacy is tested on three different DNNs. In [5], the authors can compress AlexNet to 6.9 MB without loss of accuracy, much smaller than what is achieved in this paper, due to network pruning is used [6], which removes many small-weight connections from the network. This not only compresses the network, but also reduces the complexity of the implementation. In addition, entropy coding is used in [5]. However, the quantization in [5] is fixed and not scalable. This paper focuses on the scalable quantization and adaptive bit allocation. It is also shown from Fig. 7 in [5] that pruning does not hurt quantization. Therefore the pruning and entropy coding can also be used in our scheme to further improve the performance.

Acknowledgement

This work was supported by by NSERC of Canada under grant RGPIN312262, STPGP447223, and RGPAS478109, and NVIDIA University Partnership Program.

7. REFERENCES

[1] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13:281–305, Feb. 2012.

[2] W. Chen, J. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen. Compressing neural networks with the hashing trick. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 2285–2294. JMLR Workshop and Conference Proceedings, 2015.

[3] E. L. Denton, W. Zaremba, J. Bruna, Y. Lecun, and R. Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in Neural Information Processing Systems 27*, pages 1269–1277. Curran Associates, Inc., 2014.

[4] Y. Gong, L. Liu, M. Yang, and L. Bourdev. Compressing deep convolutional networks using vector quantization. In *arXiv*, page arXiv:1412.6115, 2014.

[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] 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 28*, pages 1135–1143. 2015.

[7] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. In *International Conference on Learning Representations (ICLR)*, 2016.

[8] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. *Master's thesis, Department of Computer Science, University of Toronto*, 2009.

[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, pages 1097–1105, 2012.

[10] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[11] D. D. Lin, S. S. Talathi, and V. S. Annapureddy. Fixed point quantization of deep convolutional networks. In *arXiv*, page arXiv:1511.06393, 2015.

[12] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[13] H. Schwarz, D. Marpe, and T. Wiegand. Overview of the scalable video coding extension of the h.264/avc standard. *IEEE Trans. Circ. Syst. Video Tech.*, 17(9):1103–1120, 2007.

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

[15] G. J. Sullivan, J. M. Boyce, Y. Chen, J.-R. Ohm, C. A. Segall, and A. Vetro. Standardized extensions of high efficiency video coding. *IEEE Journal on Selected Topics in Signal Processing*, 7(6):1001–1016, Dec. 2013.

[16] D. Taubman and M. Marcellin. *JPEG 2000: image compression fundamentals, standards, and practice*. Kluwer Academic Publishers, Boston, MA, 2002.

[17] A. Vedaldi and K. Lenc. Matconvnet – convolutional neural networks for matlab. In *Proceeding of the ACM Int. Conf. on Multimedia*, 2015.