DeepSZ: A Novel Framework to Compress Deep Neural Networks by Using Error-Bounded Lossy Compression

Sian Jin
The University of Alabama
Tuscaloosa, AL, USA
sjin6@crimson.ua.edu

Sheng Di
Argonne National Laboratory
Lemont, IL, USA
sdi1@anl.gov

Xin Liang
University of California, Riverside
Riverside, CA, USA
xlian007@ucr.edu

Jiannan Tian
The University of Alabama
Tuscaloosa, AL, USA
jtian10@crimson.ua.edu

Dingwen Tao*
The University of Alabama
Tuscaloosa, AL, USA
tao@cs.ua.edu

Franck Cappello
Argonne National Laboratory
Lemont, IL, USA
cappello@mcs.anl.gov

ABSTRACT

Today's deep neural networks (DNNs) are becoming deeper and wider because of increasing demand on the analysis quality and more and more complex applications to resolve. The wide and deep DNNs, however, require large amounts of resources (such as memory, storage, and I/O), significantly restricting their utilization on resource-constrained platforms. Although some DNN simplification methods (such as weight quantization) have been proposed to address this issue, they suffer from either low compression ratios or high compression errors, which may introduce an expensive fine-tuning overhead (i.e., a costly retraining process for the target inference accuracy). In this paper, we propose *DeepSZ: an accuracy-loss expected neural network compression framework*, which involves four key steps: *network pruning*, *error bound assessment*, *optimization for error bound configuration*, and *compressed model generation*, featuring a high compression ratio and low encoding time. The contribution is threefold. (1) We develop an adaptive approach to select the feasible error bounds for each layer. (2) We build a model to estimate the overall loss of inference accuracy based on the inference accuracy degradation caused by individual decompressed layers. (3) We develop an efficient optimization algorithm to determine the best-fit configuration of error bounds in order to maximize the compression ratio under the user-set inference accuracy constraint. Experiments show that DeepSZ can compress AlexNet and VGG-16 on the ImageNet dataset by a compression ratio of 46× and 116×, respectively, and compress LeNet-300-100 and LeNet-5 on the MNIST dataset by a compression ratio of 57× and 56×, respectively, with only up to 0.3% loss of inference accuracy. Compared with other state-of-the-art methods, DeepSZ can improve the compression ratio by up to 1.43×, the DNN encoding performance by up to 4.0× with four V100 GPUs, and the decoding performance by up to 6.2×.

*Corresponding author: Dingwen Tao, Department of Computer Science, The University of Alabama, Tuscaloosa, AL 35487, USA.

HPDC '19, June 22–29, 2019, Phoenix, AZ, USA

ACM ISBN 978-1-4503-6670-0/19/06…$15.00
https://doi.org/10.1145/3307681.3326608

CCS CONCEPTS

• **Mathematics of computing → Coding theory**.

KEYWORDS

Neural Networks; Deep Learning; Lossy Compression; Performance

ACM Reference Format:
Sian Jin, Sheng Di, Xin Liang, Jiannan Tian, Dingwen Tao, and Franck Cappello. 2019. DeepSZ: A Novel Framework to Compress Deep Neural Networks by Using Error-Bounded Lossy Compression. In *The 28th International Symposium on High-Performance Parallel and Distributed Computing (HPDC '19), June 22–29, 2019, Phoenix, AZ, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3307681.3326608

1 INTRODUCTION

Deep neural networks (DNNs) have rapidly evolved to the state-of-the-art technique for many artificial intelligence tasks in various science and technology areas, for instance, image and vision recognition [35], recommender systems [45], nature language processing [7], and time series classification [48]. DNNs contain millions of parameters in an unparalleled representation, which is efficient for modeling complexity nonlinearities. Thus, using either deeper or larger DNNs can be an effective way to improve data analysis. As pointed by Wang et al. [46], the deep learning community has been acknowledging that increasing the scales of DNNs can improve the inference accuracy of image recognition tasks. A 9-layer AlexNet [21], for example, proposed by Krizhevsky et al., won the 2012 ILSVRC (ImageNet Large-Scale Visual Recognition Challenge) [23] with a top-5 accuracy of 83%. In 2014 ILSVRC, a 22-layer GoogLeNet [37] proposed by Szegedy et al. further improved the record of top-5 accuracy to 93.3%. He et al. proposed a 152-layer ResNet [18], which refreshed the record to 96.43% in 2015 ILSVRC. This trend suggests that the networks will go larger in the future.

The ever-increasing growth of networks is bringing more and more challenges to resource-limited platforms, such as mobile phones and wireless sensors. For instance, a typical use case is to train DNNs in the cloud using high-performance accelerators, such as graphic processing units (GPUs) or field-programmable gate arrays (FPGAs), and distribute the trained DNN models to edge devices for inferences [42]. According to the data released by GSMA [14], 0.8 billion users will be still using 2G networks (with a

theoretical maximum transfer speed of 1 Mbit/s) by 2020. Consequently, one practical challenge is to deliver multiple latest DNN models (i.e., an order of tens to hundreds of megabytes for each model) from cloud to edge devices through bandwidth-limited networks. An issue also arises in a wireless sensor network with DNN models, which is usually deployed in a region of interest for tracking objects or abnormal detection. Since large DNN models must be stored in external DRAM (e.g., embedded flash-based storage) and frequently fetched for inferences, they consume large amounts of energy [15]. However, sensor nodes are usually equipped with small batteries and it is unfeasible to recharge them or deploy new nodes in many cases. Therefore, how to maintain/extend the lifetime of sensor networks with large DNNs becomes challenging [47]. Compressing neural networks provides an effective way to reduce the burden of these problems. Most approaches, however, have focused on simplification methods, such as network pruning [16] and quantization [15], which suffer from limited compression quality. A straightforward idea is to leverage existing lossy compression and encoding techniques [33] to significantly improve the ratio for compressing DNNs. The existing compression strategies applied on DNNs, however, do not have error expected features, which may greatly distort the data, leading to expensive fine-tuning overhead (i.e., extra, costly retraining process).

In this paper, we propose DeepSZ: a lossy compression framework for DNNs. DeepSZ is composed of four key steps: *network pruning*, *error bound assessment*, *optimization of error bound configuration*, and *compressed model generation*. Unlike traditional compression methods used on DNNs, we perform error-bounded lossy compression on the pruned weights, an approach that can significantly reduce the data size while restricting the loss of inference accuracy. Specifically, we adapt the SZ lossy compression framework developed by us previously [11, 28, 39] to fit the context of DNN compression. In this compression framework, each data point's value would be predicted based on its neighboring data points by an adaptive, best-fit prediction method (either a Lorenzo predictor or linear regression-based predictor [28]). Then, each floating-point weight value would be converted to an integer number by a linear-scaling quantization based on the difference between the real value and predicted value and a specific error bound. Huffman encoding or other lossless compression such as Zstd [49] and Blosc [5] would be applied to significantly reduce the data size thereafter. SZ can get a much higher compression ratio on the compression of nonzero weights than other state-of-the-art compressors such as ZFP [29] can, especially because of efficient linear-scaling quantization, which contrasts with the simple vector quantization applied to the original weights in other related work [13, 15]. Moreover, our SZ compressor can control errors in more sophisticated ways, such as relative error bound and peak-signal-to-noise ratio (PSNR).

Designing an efficient lossy compression framework for DNNs raises two important challenging issues to resolve. (1) How can we determine an appropriate error bound for each layer in the neural network? Specifically, we have to explore a feasible range of error bounds for each layer, under which the lossy compression should still get a high inference accuracy for users. (2) How can we maximize the overall compression ratio regarding different layers in the DNN under user-specified loss of inference accuracy? Considering the heterogeneous and diverse data features across multiple layers, we have to explore the best-fit error bounds on the compression of different layers. A straightforward idea is to traverse all the possible error-bound combinations on different layers, which would definitely lead to an extremely high time-complexity. To address this issue, we develop a dynamic strategy to efficiently determine the best-fit error bound for each layer.

The contributions of this work are summarized as follows.

- We propose a novel, accuracy-loss expected framework, called DeepSZ, by applying our previously developed SZ error-bounded lossy compression to compress DNNs. To the best of our knowledge, this is the first attempt to do so.
- We propose an adaptive method to select the feasible range of error bounds for each layer. We also develop an effective model to estimate the overall inference accuracy loss based on the forward-propagation results with individual layers reconstructed from the error-bounded lossy compressor.
- We develop a dynamic algorithm to optimize the combined configuration regarding different layers' error bounds, significantly reducing the overall size of the neural network. In addition, because of our careful design of the accurate error control, our solution also effectively eliminates the costly retraining overhead that was generally introduced by other DNN compression methods [15, 33].
- We compare DeepSZ with two other state-of-the-art works (i.e., Deep Compression and Weightless) based on four well-known neural networks. Evaluation results demonstrate that the compression ratio of DeepSZ is 1.21×~1.43× higher than that of the other two approaches. Experiments show that DeepSZ can obtain 1.8×~4.0× encoding performance improvement on four Nvidia Tesla V100 GPUs and 4.5×~6.2× decoding performance improvement on Intel Xeon Gold 6148 CPU over the second-best approach.

The rest of the paper is organized as follows. In Section 2, we discuss the background and motivation of our research. In Section 3, we describe the design methodologies of the DeepSZ framework in detail. In Section 4, we provide a detailed analysis and comparison of DeepSZ and two other state-of-the-art approaches. In Section 5, we present the evaluation results on four well-known DNNs with multiple GPUs. In Section 6, we discuss related work. In Section 7, we summarize our conclusions and present ideas for future work.

2 BACKGROUND AND MOTIVATION

In this section, we present some background information about neural networks and lossy compression for floating-point data and discuss our research motivation.

2.1 Neural Networks

Neural networks have been widely studied and used in recent years and have produced dramatic improvements in many scientific and engineering aspects, such as computer vision [34] and natural language processing [7]. Each neural network is composed of multiprocessing layers. Among the various kinds of layers, convolutional layers and fully connected layers (denoted by `fc`-layers) have contributed the most to the recent progress in the deep learning community, especially in vision-related tasks such as image classification and object detection. One convolutional layer consists of a

Table 1: Architectures of example nerual networks.

Neural Networks	LeNet-300-100	LeNet-5	AlexNet	VGG-16
conv layers	0	3	5	13
fc-layers	3	2	3	3
ip1/fc6	300 × 784	500 × 800	4096 × 9216	4096 × 25088
ip2/fc7	100 × 300	10 × 500	4096 × 4096	4096 × 4096
ip3/fc8	10 × 100	-	1000 × 4096	1000 × 4096
conv fwd time	0 ms	0.5 ms	116.5 ms	149.8 ms
fc fwd time	0.30 ms	0.12 ms	2.5 ms	1.7 ms
total size	1.1 MB	1.7 MB	243.9 MB	553.4 MB
fc-layers' size (%)	100%	95.3%	96.1%	89.4%

set of filters that slide in the input dataset and perform convolution with the signal in the sliding window. `fc`-layers are connected by a dense weight matrix and forward the signals by a matrix-matrix multiplication. The filters in the convolutional layers and the weight matrices in the `fc`-layers dominate the storage space of the neural networks, which will become larger as the networks become deeper or wider. In neural networks, the forward pass refers to the calculation process, which traverses from the first layer to the last layer. The backward pass refers to the process to update the weights by stochastic gradient descent, which traverses from the last layer backward to the first layer. During the training period, both forward and backward passes are performed, whereas only the forward pass is performed for testing. In the following discussion, TEST refers to the forward pass process on the test dataset for generating the inference accuracy of the neural network.

According to prior studies [3], although convolutional layers occupy most of the computation time (~95%) because of the expensive convolution operations, they take up little storage space (~5%). On the other hand, `fc`-layers require large storage space (~95%) because of the large dense matrices, while consuming little computation time (~5%). This phenomenon is also verified in our experiments. For demonstration purposes, we present the breakdown of storage and computational overhead for four well-known networks. As shown in Table 1, the `fc`-layers take the majority of the networks' storage space (i.e., 89.4% ~ 96.1%) in all three cases; however, they have much lower computational cost (i.e., about 1% ~ 2% for VGG-16 and AlexNet and 20% for LeNet-5) than the convolutional layers do. Hence, we are motivated to leverage lossy compression techniques in order to trade computation time for storage space on the `fc`-layers in resource-constrained scenarios, our aim is to significantly reduce the storage requirements of neural networks while introducing little computation overhead.

2.2 Lossy Compression for Floating-Point Data

Floating-point data compression has been studied for decades. The data compressors can be split into two categories: lossless and lossy. Lossless compressors such as GZIP [10], FPZIP [31], and BlosC [2] cannot significantly reduce the floating-point data size because of the significant randomness of the ending mantissa bits. The compression ratios of lossless compression are generally limited to 2:1, according to recent studies [30, 36].

Many lossy compressors supporting floating-point data were proposed originally for visualization. Hence, many lossy compressors employ the techniques directly inherited from lossy compression of images, such as variations of wavelet transforms, coefficient prioritization, and vector quantization. Lossy compressors for image processing are designed and optimized considering human perception, such as JPEG2000 [41]. While such compressors may be adequate for scientific visualization, they do not provide pointwise error controls on demand. For example, most lossy compressors designed for visualization do not provide control of a global upper bound on the compression error (the maximum compression error, or L_{inf} norm of the compression error).

A new generation of lossy compression techniques for floating-point data has been developed recently. SZ, ZFP, and MGARD [1] are three typical error-bounded compressors. SZ [11, 28, 39] predicts each data point's value by its neighboring data points in a multidimensional space with an adaptive predictor (using either a Lorenzo predictor [19] or linear regression [28]). Next, it performs an error-controlled linear-scaling quantization to convert all floating-point values to an array of integer numbers. And then it performs a customized Huffman coding and lossless compression to shrink the data size significantly. ZFP [29] splits the whole dataset into many small blocks and compresses the data in each block separately by four steps: alignment of exponent, orthogonal transform, fixed-point integer conversion, and bit-plane-based embedded coding. MGARD uses multigrid methods to compress multidimensional floating-point data [1]. Many independent studies [11, 28, 32, 38, 39] have showed that SZ outperforms the other two compressors in terms of compression ratio, especially on 1-D floating-point datasets; note that the datasets to compress in our case are 1-D floating-point arrays after conversion.

Today's lossy compression techniques have been used in HPC scientific applications for saving storage space and reducing the I/O cost of saving data. However, *how to effectively and efficiently utilize error-bounded lossy compressors to significantly reduce the neural network size and encoding time, while still maintaining a high inference accuracy*, remains an open question.

3 DESIGN METHODOLOGIES

In this section, we describe in detail DEEPSZ, our proposed lossy compression framework for neural networks.

3.1 Overview of DEEPSZ Framework

The general workflow of the DEEPSZ framework is presented in Figure 1. As illustrated in the figure, DEEPSZ consists of four key steps: *network pruning*, *error bound assessment*, *optimization of the error bound configuration*, and *generation of the compressed model*. The first step is to adopt network pruning in order to reduce the network complexity and mitigate the overfitting problem caused by the large number of parameters in the network. The second step is to apply the error-bounded lossy compression to the pruned `fc`-layers and assess the impacts of different error bounds on the inference accuracy for different `fc`-layers. Based on the inference accuracy degradation, DEEPSZ will identify the feasible range of error bounds for each `fc`-layer and collect the results of inference accuracy degradation and compressed layer size based on these bounds. This step can effectively narrow the range of the best-fit

[1] Some compressors such as ISABELA [22] were designed with pointwise error control, but tests [11] have shown that the maximum error could be much larger than the user-set error bound.

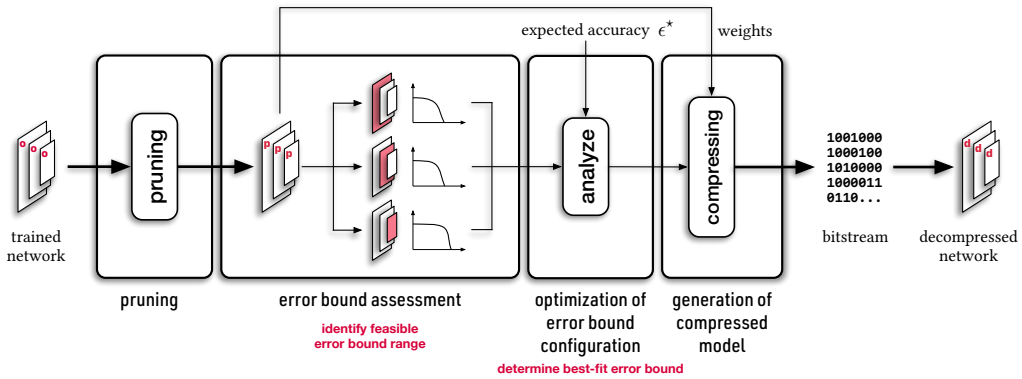


Figure 1: Overview of DeepSZ framework for neural network compression.

error bounds for each layer. Note that we focus only on the fc-layers in this work because it dominates most of the storage space, as discussed in Section 2. The third step is to determine the best-fit error bound for each fc-layer based on the narrowed feasible range generated from the second step. DeepSZ will compress the network as much as possible while satisfying the user-set inference accuracy requirement. The fourth step is to generate the compressed network based on the optimized error bounds and best-fit lossless compressor. In the remainder of this section, we discuss each step of DeepSZ in detail.

3.2 Network Pruning

An fc-layer in DNNs can be represented by a floating-point matrix. Each nonzero element in the matrix represents the weight of one connection between previous layer and current layer. Previous studies on modern neural network models [17, 27] have shown that most of the weights in fc-layers are redundant and can be pruned without any impact on the inference accuracy. Moreover, network pruning is an effective way to prevent the DNN model from overfitting.

We build our pruning method on top of prior state-of-the-art techniques [15], which can prune DNNs without loss of inference accuracy. We first set up thresholds for each fc-layer and prune their weights based on these thresholds: every weight below these thresholds will be removed. The thresholds are set based on the predefined pruning ratios suggested by previous studies [16]. Then, we retrain the network with masks (i.e., zero weights are marked as unchanged) on fc-layers such that the weights that have been pruned can be kept zero. This pruning method is called *magnitude threshold plus retraining* (Magnitude). Note that this process can start from well-trained networks; more details can be found in [16]. We note that Reagen et al. [33] presented another pruning method, dynamic network surgery (DNS), and evaluated its performance on several networks. The time overhead of DNS applied to large networks (such as VGG-16) is very high, however, because DNS needs to iteratively prune the weights and retrain the network based on increasing thresholds. In contrast, Magnitude has relatively lower time overhead thus can be well applied to large neural networks. Therefore, we focus on the Magnitude method in this paper.

After the network pruning, the weight matrix becomes sparse, so it can be represented by a sparse matrix format, such as compressed sparse row (CSR) or compressed sparse column (CSC) format. Unlike the traditional format that uses three 1-D arrays (e.g., arrays for nonzero values, the extents of rows, and column indices in CSR), we only use two 1-D arrays to represent one fc-layer after the pruning. One array is named data array [2]; it is used to store the floating-point weights (32 bits per value); the other one is named index array; it is used to store the index differences between two consequent nonzero weights (8 bits per value). Similar to [15], if the index difference exceeds 256 (i.e., 2^8), we additionally save a zero padding to data array and 255 to index array. Here we use sparse matrix representation because the inference accuracy can be dropped sharply (i.e., to 20% on the tested networks) if the lossy compression is applied to the matrices of pruned weights (i.e., 2-D arrays) based on our experiments. Note that the real compression ratio after the pruning step (i.e., original size divided by the CSR size) is always lower than the compression ratio that we set for the pruning (i.e., one divided by the pruning ratio), because after the pruning every nonzero weight will be represented by 40 bits (8 for index and 32 for data), which is slightly larger than the original 32 bits. Based on our evaluation results, the pruning step can typically reduce the size of fc-layers by about 8× to 20× if the pruning ratio is set to be around 4% to 10%.

3.3 Error Bound Assessment

SZ lossy compression usually has a much higher compression ratio on 1-D datasets than other lossy compression methods do [11, 38]. Our floating-point datasets are 1-D data arrays, as described in Section 3.2. For demonstration purposes, we evaluated SZ and ZFP[3] on the 1-D data arrays of each fc-layer in AlexNet and VGG-16. The compression ratios are presented in Figure 2. The figure shows that SZ consistently outperforms ZFP in terms of compression ratios on the tested fc-layers with absolute error bounds of 10^{-2}, 10^{-3}, and 10^{-4}. Although SZ has higher compression and decompression times than does ZFP [39], they are still much lower than the time overheads of forward or backward pass in neural networks. Taking these facts into consideration, we propose to use SZ lossy compression in our DeepSZ framework.

Error-bounded lossy compression can provide high compression ratio but can also bring bounded errors to neural networks, leading to possible loss of inference accuracy. Thus, before adopting SZ to compress fc-layers, we need to find the best-fit error bound

[2] We note that the nonzero floating-point weights will be condensed into a 1-D array or linked list regardless of the sparse matrix representation format.
[3] Many recent studies, such as [40], have demonstrated that SZ and ZFP are two leading lossy compressors for floating-point data.

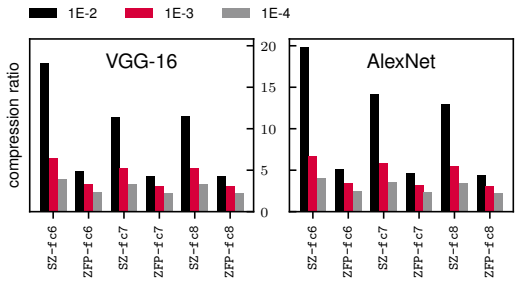


Figure 2: Compression ratios of SZ and ZFP lossy compression on fc-layer in AlexNet and VGG-16.

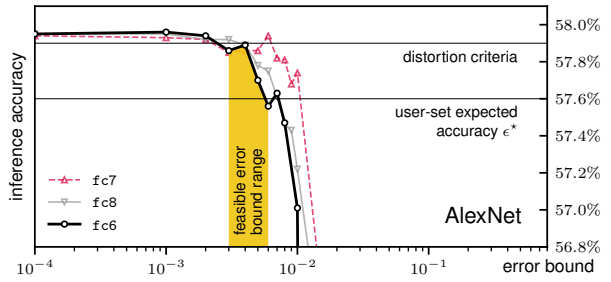


Figure 3: Inference accuracy of different error bounds on the fc-layers in AlexNet.

for each layer. Our idea is to narrow the best-fit error bounds by identifying a feasible error bound range for each fc-layer with a high compression ratio and also bounded loss of inference accuracy (detailed in this subsection) and then fine tune the best-fit error bound within the range (detailed in next subsection). To this end, we need to understand the impact of different error bounds of each fc-layer on the overall inference accuracy. Specifically, we use SZ to compress each fc-layer's DATA ARRAY with different error bounds and use its decompressed array to reconstruct the fc-layer while leaving the other fc-layers uncompressed or unchanged. Based on the reconstructed network (only one fc-layer is modified), we can perform the forward pass on the test data to generate the inference accuracy. Thus, we can get a series of inference accuracies based on different error bounds for each fc-layer in the network. For example, Figure 3 presents the accuracies based on the absolute error bounds from 10^{-1} to 10^{-4} for the three fc-layers in AlexNet. Note that in order to reduce the time overhead, we choose only a group of error bounds for compression, decompression, and checking inference accuracy rather than all the error bounds shown in Figure 3. In the following text we will describe how to determine the error bounds.

In our solution, we test the inference accuracy with only one compressed layer in every test, instead of using a brute-force method to search all possible test cases (i.e., all combinations of error bounds across all layers), for the following two reasons. (1) We observe that the fc-layers in neural networks usually have *independent* characteristics in the context of SZ lossy compression. That is, two reconstructed fc-layers based on SZ affect the overall accuracy independently. Thus, the overall loss of inference accuracy can be composed of (and thus estimated by) the losses of inference accuracy introduced by individual layers. We will discuss more details in Section 3.4. (2) Checking the inference accuracy in a test with only one reconstructed layer using multiple error bounds has much

Algorithm 1 Error bound assessment for fc-layers in deep neural networks.

Notation: layer: ℓ; ACCURACY DEGRADATION: Δ; error bound: eb; threshold: τ; Expected Accuracy Loss: $\epsilon^\star$; the given network: $\mathcal{N}$; size of compressed layer ℓ with eb: $\sigma_{(\ell;eb)}$

Global: $\mathcal{N}$, $\epsilon^\star$, all $\sigma_{(\ell;eb)}$, all $\Delta_{(\ell;eb)}$

```
1  procedure CHECK(ℓ°, eb°, base)
2      test Δ_(ℓ°;eb°) on N and update σ_(ℓ°;eb°)
3      if Δ_(ℓ°;eb°) > ε* then
4          terminate the CHECK
5      else
6          eb° ← eb°+base
7          base ← 10 × base if eb° = 10 × base
8          CHECK(ℓ°, eb°, base)
9      end if
10 end procedure
11
12 procedure ERRORBOUNDASSESSMENT
13     for ℓ ← layers in network do
14         for β ← [1E-3,1E-2,1E-1] do          ▷ can be pushed to 1E-4 etc.
15             if Δ_(ℓ;β) > 0.1% then
16                 CHECK(ℓ, 1 × β/10, β/10)
17                 break
18             end if
19         end for
20         return all σ_(ℓ;eb), all Δ_(ℓ;eb)
21     end for
22 end procedure
```

less computational cost than does a brute-force method involving every possible combination of error bounds across multiple layers. Specifically, our solution has a linear time complexity compared with the brute-force method with an exponential time complexity. VGG-16, for instance, has three fc-layers. Assume we have 10 candidate error bounds for each layer. Then the brute-force method needs to check 1,000 test cases, each involving one compression, one decompression, and one forward-pass test. By comparison, our solution has only 30 test cases to check, thus reducing the testing time to 3% compared with the brute-force method.

We propose an algorithm to identify the feasible range of error bounds and collect the results about inference accuracy degradation and compressed layer size based on these bounds for each fc-layer. We present the pseudo-code in Algorithm 1. The inputs of the algorithm include the architecture of the neural network, the pruned weights of the network, and the user-set loss of inference accuracy. Lines 12–21 show the main loop of this algorithm. Specifically, when the loss of inference accuracy exceeds a criterion of 0.1% (called distortion criterion) in terms of absolute percentage, we treat the reconstructed network to be distorted. We search the feasible range of error bounds by checking the accuracy based on multiple error bounds. The error bound to check starts from a certain value (i.e., the default value is 10^{-3}; the reason will be discussed in Section 5.1) and will be increased by an order of magnitude (10×) each time. As the accuracy drops below the criterion (i.e., 0.1%) at a certain error bound eb', we set the starting point of the range to be $eb'/10$. Note that the default value of 10^{-3} can be further decreased, such as 10^{-4}, based on different neural networks.

From Lines 1–10, we determine the ending point of the range based on the accuracy of reconstructed network. The ending point is the first error bound that the accuracy drops below the user's expected accuracy ϵ^*. Once the feasible range is generated (as shown in Figure 3), we conduct the tests with the error bounds in the feasible range and collect the sizes of compressed layer and accuracy degradation results. Lines 6–8 describe how we choose

the error bounds within the range. For example, if the range is $[8 \times 10^{-3}, 3 \times 10^{-2}]$, we test the error bounds of 8×10^{-3}, 9×10^{-3}, 10^{-2}, 2×10^{-3}, and 3×10^{-3}.

3.4 Optimization of Error Bound Configuration

Depending on the number of input and output neurons, the sizes of different fc-layers vary dramatically. For example, the largest fc-layer of VGG-16 is fc6 (i.e., 25,088×4,096), which is 25× larger than the smallest layer fc8 (i.e., 1,000×4,096). Compressing larger fc-layers with a higher error bound can lead to a higher compression ratio, which can benefit the overall ratio. However, the higher error bound also brings more errors to the network, which may degrade the overall accuracy in turn. Therefore, how to determine the optimal error bound for each fc-layer is an important problem.

From our experiments we discovered that the overall accuracy loss in the neural network exhibits an approximate linearity in terms of the accuracy degraded in each fc-layer in the context of SZ lossy compression when the targeted loss of inference accuracy is lower than 2% based on our tested neural networks. In other words, the overall accuracy loss $\Lambda^\star$ is approximately equal to the sum of each layer's accuracy degradation Δ_ℓ, as shown in Equation (1), where eb_ℓ is a given arbitrary error bound for each fc-layer ℓ:

$$\Delta^\star = \sum \Delta_{\ell, eb_\ell}, \qquad \Delta^\star < 2\%. \tag{1}$$

This observation can be explained by a theoretical analysis on the independent impact of compression error introduced in each fc-layer on the output of a neural network. Assume t_1 and t_2 to be the original outputs of two successive fc-layers. Assume ΔW_1 and ΔW_2 to be the compression errors introduced to the weights of these two fc-layers, respetively. When computations pass these two fc-layers, the outputs of the two fc-layer would be

$$t_1^\star = f((W_1 + \Delta W_1)t_0 + b_1), \quad t_2^\star = f((W_2 + \Delta W_2)t_1^\star + b_2), \tag{2}$$

where f represents the activation function of fc-layers, W represents the weights, and b represents the bias. For most of neural networks, the activation function f used in fc-layers is rectified linear unit (a.k.a., ReLU), which is $max(0, x)$ (where x is the input to a neuron). In this case, due to $\Delta W_1 << W_1$ and $\Delta W_2 << W_2$, Equation (2) can be simplified to

$$t_1^\star = f(W_1 t_0 + b_1) + f\Delta W_1 t_0 = t_1 + f\Delta W_1 t_0, \tag{3}$$

$$t_2^\star = t_2 + f W_2 \Delta W_1 t_0 + f\Delta W_2 t_1. \tag{4}$$

The final output $T^\star$ of the network would be

$$T^\star = T + f\Delta W_1 t_0 \Pi_{i=2}^n (W_i) + f\Delta W_2 t_1 \Pi_{i=3}^n (W_i). \tag{5}$$

where n is the total number of fc-layers and T is the original output. We can clearly observe that two errors ΔW_1 and ΔW_2 have independent impacts on the final output $T^\star$. Therefore, we can conclude that compression error introduced in each fc-layer would have independent impact on final network's output. In order to assure $\Delta W << W$, we will choose the error bound to be lower than 0.1 in our experiments. Finally, the relationship between final output and accuracy loss is approximately linear, which will be experimentally demonstrated in Section 5.2.

We propose Algorithm 2 to determine the best-fit error bound for each layer. The inputs include the accuracy degradation and compressed size of each fc-layer based on our tested error bounds,

Algorithm 2 Optimization of Error Bound Configuration

Notation: layer: ℓ; error bound: eb; accuracy: ϵ; AccuracyDegradation at ℓ with eb: $\Delta_{(\ell;eb)}$; size: σ; TotalSize: S; ExpectAccuracy: $\epsilon^\star$
Input: $\Delta_{(\ell;eb)}$, $\epsilon^\star$, $\sigma_{(\ell;eb)}$
Output: eb_ℓ

1 **procedure** OptimizeErrorBound
2 $S_{(\ell;\Delta)} \leftarrow$ maximum
3 $S_{(\ell_{\text{zero}};\epsilon)} \leftarrow$ zero
4 **for** $\ell \leftarrow$ layers in network **do**
5 **for** $eb \leftarrow$ tested error bounds **do**
6 **for** $\epsilon \leftarrow [0 \ldots 100] \times \epsilon^\star$ **do**
7 **if** $S_{(\ell_{\text{prev}};\epsilon)} + \sigma_{(\ell;eb)} < S_{\left(\ell;\epsilon+\Delta_{(\ell,eb)}\right)}$ **then**
8 $S_{\left(\ell;\epsilon+\Delta_{(\ell,eb)}\right)} \leftarrow S_{(\ell_{\text{prev}};\epsilon)} + \sigma_{(\ell;eb)}$
9 **end if**
10 **end for**
11 **end for**
12 **end for**
13 let $\ell^\star$ be the final layer in the network
14 find minimum $S_{(\ell^\star;\Delta)}$
15 **for** $\ell \leftarrow$ layers in network **do** ▷ tracing back
16 **for** $eb \leftarrow$ tested error bound **do**
17 $eb_\ell \leftarrow eb$ **if** $S_{(\ell;\epsilon)} - \sigma_{(\ell;eb)} = S_{\left(\ell_{\text{prev}};\epsilon-\Delta_{(\ell;eb)}\right)}$
18 **end for**
19 **end for**
20 **end procedure**

which are outputted by our previous error bound assessment (Section 3.3) and an expected loss of accuracy set by users. Algorithm 2 can minimize the total size of the compressed fc-layers while ensuring the sum of the accuracy degradation of each layer to be within the expected accuracy loss. Specifically, the first part of Algorithm 2 (Lines 2–14) finds the minimum total size of compressed fc-layers with different combinations of error bounds before a certain layer by using a variation of Knapsack algorithm. Then, the algorithm traces back to determine the error bound for each fc-layer (Lines 15–19). More specifically, we save the minimal size of all fc-layers (before ℓ-layer) and the accuracy loss of ϵ^* to the variable $S_{(\ell;\epsilon)}$. After we find the minimum compressed size of all the fc-layers under the constraint of the overall accuracy loss (Lines 13–14), we trace back from the minimal $S_{(\ell;\epsilon)}$ to identify the error bound combination for each layer (Lines 15–19).

Besides this optimization of the compression ratio by an expected accuracy loss (i.e., expected-accuracy mode), DeepSZ can optimize the overall accuracy with an expected compression ratio (i.e., expected-ratio mode).The algorithm of the fixed-rate mode is similar to Algorithm 2 but just reverses the compressed size and accuracy degradation. Based on these two modes, we can fine tune the balance between accuracy loss and compression ratio for a neural network, which is much more flexible than other methods.

3.5 Generation of Compressed Model

The last step in our framework is to generate the compressed model by using SZ lossy compression on the data arrays with the error bounds (obtained in Step 3) and the best-fit lossless compression on the index arrays. The index array represents the locations of nonzero weights, which need to be compressed losslessly. DeepSZ provides three state-of-the-art lossless compressors: Gzip [10], Zstandard [49], and Blosc [49]. More lossless compressors can be integrated into the framework in the future. In our experiments, we identified that Zstandard always leads to the highest compression ratio compared with the other two compressors, as shown in

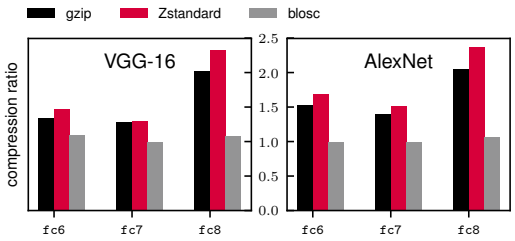


Figure 4: Compression ratios of different layers' INDEX ARRAYS with different lossless compressors on AlexNet and VGG-16.

Figure 4. After these four steps of DEEPSZ, the compressed neural network model is generated. In this paper, we use ENCODING to refer this whole process of generating compressed DNNs and DECODING to refer the process of reconstructing DNNs.

Once the network is needed for forward pass, it must be decoded. During the decoding, DEEPSZ will decompress the DATA ARRAYS using the SZ lossy compression and the INDEX ARRAYS using the best-fit lossless compression (e.g., Zstandard). Then, the sparse matrix can be reconstructed based on the decompressed DATA ARRAY and INDEX ARRAY for each `fc`-layer. Finally, the whole neural network can be decoded. Note that the computational cost of the decoding in DEEPSZ is relatively low compared to that of the forward pass with a batch of images. We will analyze the performance overhead of our decoding in detail and compare it with other state-of-the-art methods next section.

4 DETAILED ANALYSIS AND COMPARISON

In this section, we analyze DEEPSZ in detail and compare it with two other state-of-the-art solutions: Weightless [33] and Deep Compression [15]. Our analyses focus on both performance and storage.

4.1 Performance Analysis of DEEPSZ

For Algorithm 1 in DEEPSZ, the computational cost is focused mostly on performing the tests with different error bounds to check the corresponding accuracies, while compression and decompression both cost negligible time overhead. Let us take AlexNet as an example. Compressing and decompressing one DATA ARRAY (about tens of megabytes per `fc`-layer's DATA ARRAY, as shown in Table 2c), and reconstructing the network based on the decompressed layer typically take no more than one second on an Nvidia Tesla V100 GPU,[4] whereas testing the reconstructed network with 50,000 images in the ImageNet dataset will take about 55 seconds (10 seconds for data transfer, 5 seconds for initialization, and 40 seconds for forward computations). In this case, DEEPSZ needs to perform 12 tests on each `fc`-layer, - bringing the total to 36 tests. In contrast, the 36 tests require performing forward passes of 1.8 million images considering 50,000 images in the ImageNet test data. Based on our experiments, the execution time of one epoch is about 42 times higher than that of one TEST with an Nvidia V100 GPU on AlexNet [5]. Thus, the workload of 1.8 million images in the TEST is equivalent to training about $\frac{6}{7}$ epochs of data (i.e., one TEST is about $\frac{1}{42}$ epochs). Therefore, the time complexity of Algorithm 1 is $O(\frac{c \cdot k \cdot M}{42})$, where c is the number of tests per layer (e.g., 12 for AlexNet), k is the number of `fc`-layers (e.g., 3 for AlexNet), and $O(M)$ is the time complexity for training one epoch of data. For AlexNet, for example, we can set c to 12 and k to 3; hence, the time complexity is about $O(\frac{6 \cdot M}{7})$.

For Algorithm 2 in DEEPSZ, because of our optimization in Algorithm 1, the input dimension of the algorithm is very small (i.e., $O(c \cdot k)$, for example 36 pairs of inference accuracy degradation and compressed size for 3-`fc`-layer AlexNet. Based on the time complexity of the Knapsack algorithm, the time complexity of Algorithm 2 is $O(100 \cdot \epsilon^{\star} \cdot c \cdot k)$. We note that $O(100 \cdot \epsilon^{\star} \cdot c \cdot k)$ is far smaller than $O(\frac{c \cdot k \cdot M}{42})$ because $O(\frac{M}{42})$ is much larger than $O(100 \cdot \epsilon)$. Thus, the computational cost of Algorithm 2 would be relatively small compared with multiple tests of inference accuracy in Step 2. Overall, we conclude that the time complexity of DEEPSZ's encoding is $O(\frac{c \cdot k \cdot M}{42})$, which is much less than that of traditional methods with retraining (typically $O(5 \cdot M)$ to $O(10 \cdot M)$ in the ImageNet dataset). It is worth noting that the scalability of TEST (i.e., embarrassing parallelism) is higher than that of training in parallel, thus, the time complexity of DEEPSZ compared with training will be further reduced with increasing scale.

For DEEPSZ's decoding, the computational cost is also comparatively low because it performs an $O(n_{\text{pruned}})$ lossy decompression with SZ, an $O(n_{\text{pruned}})$ lossless decompression with the best-fit lossless compressor (e.g., Zstandard) and an $O(n)$ sparse-dense matrix conversion. Here we denote the number of pruned weights by n_{pruned} and the number of original weights by n. Overall, the time complexity of DEEPSZ's decoding is $O(n)$.

4.2 Comparison with Weightless

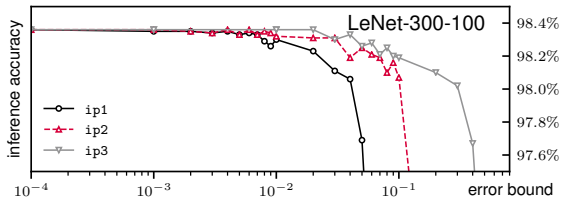
DEEPSZ has four major advantages over Weightless. (1) Weightless has higher time overhead than does DEEPSZ for encoding. After Weightless reconstructs the layer based on the Bloomier filter, the inference accuracy can drop dramatically. For example, the inference accuracy drops about 3% when compressing `fc6` in VGG-16 using Weightless. Thus, Weightless requires retraining the other layers to recover the overall inference accuracy, whereas DEEPSZ does not require any retraining. (2) Weightless has higher time overhead than does DEEPSZ on decoding. To decode one element, Weightless has to calculate four hash functions based on all the values (including zero values) in the pruned matrix and check the hash table to determine the value of this element, leading to much higher time overhead compared with DEEPSZ. (3) Weightless can compress only one layer (usually the largest layer). By contrast, DEEPSZ can compress all `fc`-layers, leading to higher overall compression ratio. (4) DEEPSZ provides two modes to users. Even for the fixed-accuracy mode, users can set an expected loss of inference accuracy in DEEPSZ and get as high a compression ratio as possible, whereas Weightless is unable to provide such flexibility.

4.3 Comparison with Deep Compression

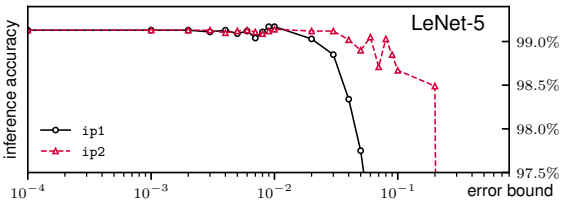
Similar to Weightless, Deep Compression also requires retraining the whole network to mitigate the inference accuracy loss caused by its quantization. Deep Compression adopts a simple quantization technique on the pruned weights. It quantizes all the nonzero

[4] Based on a recent study [39], SZ's compression and decompression rate are about 80 MB/sec and 150 MB/sec, respectively, with the error bound of 10^{-3} on a 2.3 GHz Intel Core i7 processor.

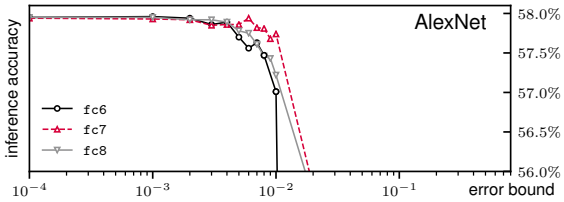
[5] One epoch contains one forward pass and one backward pass of the 1.28 million images in the training dataset and takes about 15.7 minutes for AlexNet on a single Nvidia Tesla V100 GPU, based on our experiments.



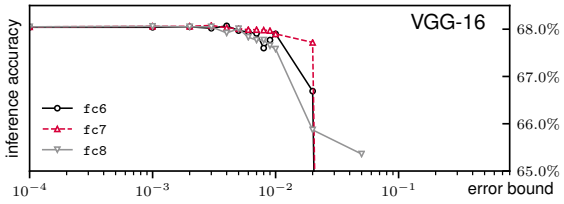
(a) LeNet-300-100



(b) LeNet-5



(c) AlexNet



(d) VGG-16

Figure 5: Inference accuracy of different error bounds on the fc-layers in LeNet300-100, LeNet-5, AlexNet, and VGG-16

weights to a group of floating-point values based on a code-book. The number of these values in the code-book is always 2^k, where k refers to the number of bits used to represent one weight. Using 5 bits per weight, for example, can map every nonzero weights to a 32-value code-book. Unlike Deep Compression applying a simple quantization to the weights, DeepSZ applies an error-bounded linear-scaling quantization to the difference between the predicted weight and real weight based on a best-fit prediction method, leading to higher compression ratios and fine-granularity error controls. Similar to Weightless, Deep Compression has lower flexibility than DeepSZ has in terms of the balance between the ratio and inference accuracy. Since the number of floating-point values the code-book can represent is always 2^k, the inference accuracy under Deep Compression may drop significantly (shown in Section 5.2) with increasing compression ratios (or lower bit rates), leading to unbounded inference accuracy.

5 EXPERIMENTAL EVALUATION

In this section, we evaluate our proposed DeepSZ framework by comparing it with state-of-the-art methods.

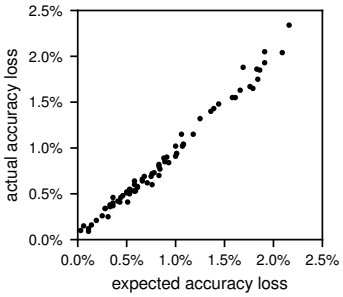


Figure 6: Approximate linearity relationships of error introduced by different fc-layers

5.1 Experimental Setting

We conduct our evaluation on a single core of an MacBook Pro with Intel Core i7-8750H Processors (with 32 GB of memory) and parallel experiments using four Nvidia Tesla V100 GPUs (each with 16 GB of memory) on the node of the Pantarhei cluster at the University of Alabama. The four GPUs are connected via NVLink [12]. We implement DeepSZ based on the Caffe deep learning framework [20] (v1.0) and SZ lossy compression library (v2.0) [28]. We evaluate DeepSZ on four well-known neural networks: LeNet-300-100 [25], LeNet-5 [24], AlexNet [21], and VGG-16 [35]. We train/test LeNet300-100 and LeNet-5 on the MNIST dataset and AlexNet and VGG-16 on the ImageNet dataset [6], respectively. These neural networks and datasets are commonly used in evaluation studies [15, 33]. We present the details of their architectures in Table 1. Note that the fc-layers occupy most of the storage space (i.e., 89.4% ~ 96.1%). We use the default solver (i.e., stochastic gradient descent (SGD)) in Caffe for all training. We set the expected loss of inference accuracy to 0.2% for two LeNets and 0.4% for AlexNet and VGG-16, without loss of generality. We also set the expected loss of inference accuracy to zero and demonstrate the flexibility of DeepSZ.

We note that in an fc-layer of a neural network, weights are floating-point numbers between -1.0 and 1.0; more generally, for a trained network, such as AlexNet and VGG-16, the value ranges of their weights are typically between -0.3 and +0.3. Thus, the absolute error bounds in the order of 10^{-1} are relatively large compared with the weight values. Consequently, using the error bounds in the order of 10^{-1} would significantly affect the overall inference accuracy (i.e., dropped to less than 20%), as illustrated in Figure 5. We also note that the absolute error bound of 10^{-4} can maintain the inference accuracy without any loss for these networks. Thus, we set 10^{-3} to be the default value for initial point of the error bound to be checked. It is worth pointing out that the output of a neural network for image recognition is a vector of probabilities for different types. Based on these probabilities, a neural network can predict the type of an image. Thus, the inference accuracy in the context of image recognition and neural network is the precision ratio. Similar to previous studies [15, 33, 43], no recall ratio and F1 score would be measured in our experiments.

5.2 Evaluation Results

5.2.1 Linearity of Accuracy Loss. We first experimentally demonstrate the approximate linearity of accuracy loss on fc-layers based on AlexNet and VGG-16. We set different combinations of error

[6] ImageNet Large Scale Visual Recognition Challenge 2012 (ILSVRC2012)

Table 2: fc-layers' compression statistics for 4 NNs

(a) fc-layers' compressing statistics for LeNet-300-100

Layer	Original Size	Pruning Ratio	CSR Size	DeepSZ Compressed
ip1	941 KB	8%	94 KB	15.2 KB
ip2	120 KB	9%	14 KB	1.6 KB
ip3	4 KB	26%	1.3 KB	0.7 KB
overall	1056 KB	8.25%	109 KB (9.7 ×)	19.1 KB (55.8 ×)

(b) fc-layers' compressing statistics for LeNet-5

Layer	Original Size	Pruning Ratio	CSR Size	DeepSZ Compressed
ip1	1600 KB	8%	160 KB	27.3 KB
ip2	20 KB	19%	4.8 KB	0.93 KB
overall	1620 KB	8.1%	165 KB (9.8 ×)	28.27 KB (57.3 ×)

(c) fc-layers' compressing statistics for AlexNet

Layer	Original Size	Pruning Ratio	CSR size	DeepSZ Compressed
fc6	151.0 MB	9%	17.0 MB	2.77 MB
fc7	67.1 MB	9%	7.5 MB	1.44 MB
fc8	16.4 MB	25%	5.1 MB	0.94 MB
overall	234.5 MB	10.1%	29.6 MB (7.9 ×)	5.15 MB (45.5 ×)

(d) fc-layers' compressing statistics for VGG-16

Layer	Original Size	Pruning Ratio	CSR Size	DeepSZ Compressed
fc6	411.0 MB	3%	15.4 MB	2.70 MB
fc7	67.1 MB	4%	3.4 MB	0.75 MB
fc8	16.4 MB	24%	4.8 MB	0.83 MB
overall	494.5 MB	3.8%	23.6 MB (20.9 ×)	4.28 MB (115.6 ×)

bounds (i.e., within 0.1) for fc-layers and compare our expected accuracy loss (based on Equation (1)) with the actual accuracy loss, as shown in Figure 6. Specifically, x-axis shows our expected accuracy loss, which is the sum of accuracy degradation in each fc-layer, and y-axis shows the actual accuracy loss. We can observe a clear linear relationship when the overall accuracy loss is lower than 2%. Therefore, we can use this approximate linearity of accuracy loss to perform the following optimization of error bound configurations discussed in Section 3.4.

5.2.2 Compression Ratio. We then present the experimental results of DeepSZ in terms of compression ratio and compare with Deep Compression and Weightless.

LeNet-300-100 and LeNet-5 on MNIST. First, we evaluate DeepSZ on LeNet300-100 and LeNet-5 with the MNIST dataset [26]. LeNet-300-100 contains only three fc-layers (i.e., ip1, ip2, and ip3). LeNet-5 contains three convolutional layers and two fc-layers (i.e., ip1 and ip2). The fc-layers dominate: 100% and 95.3% of the overall sizes of LeNet-300-100 and LeNet-5, respectively, as shown in Table 1. DeepSZ first prunes the network with the pruning ratios suggested by [15] (as shown in Table 2a and 2b) and stores the pruned weights in the data arrays and index arrays. After the pruning step, LeNet-300-100 and LeNet-5 can be reduced by 9.7× and 9.8×, respectively. Note that the compression ratio is slightly different from the pruning ratio because every nonzero pruned weight requires 40 bits instead of 32 bits, as discussed in Section 3.2. Then, DeepSZ deploys the error bound assessment step to the pruned network and gets the feasible ranges of error bounds for fc-layers. The feasible ranges are $[10^{-2}, 3\times10^{-2}]$, $[10^{-2}, 8\times10^{-2}]$, and $[10^{-2}, 2\times10^{-1}]$ for ip1, ip2, and ip3 of LeNet-300-100, respectively, as shown in Figure 5a. The ranges are $[10^{-2}, 3\times10^{-2}]$ and $[10^{-2}, 9\times10^{-2}]$ for ip1 and ip2 of LeNet-5, respectively, as shown in Figure 5b. DeepSZ then optimizes the configuration of the error bounds based on Algorithm 2. The final error bounds of ip1, ip2, and ip3 of LeNet-300-100 are 2×10^{-2}, 3×10^{-2}, and 4×10^{-2}, respectively. The final error bounds of ip1 and ip2 of LeNet-5 are 3×10^{-2} and 8×10^{-2}, respectively. DeepSZ then adopts the best-fit lossless compressor—Zstandard—to compress the index arrays. As shown in Table 2a and 2b, DeepSZ can compress fc-layers of LeNet-300-100 by 55.8× and the fc-layers of LeNet-5 by 57.3× with *no loss of inference accuracy*. We note that each fc-layer has a threshold of error bound, after which the inference accuracy begins to drop sharply. This phenomenon is also true for other networks, such as AlexNet and VGG-16. It demonstrates that how to determine the proper error bounds for each fc-layer is a critical problem, for which DeepSZ provides an efficient, fine-tuning solution (see Section 3.3 and Section 3.4).

AlexNet on ImageNet. We next evaluate DeepSZ on a much larger network, AlexNet, with the ImageNet dataset [21]. AlexNet contains five convolutional layers and three fc-layers (i.e., fc6, fc7, and fc8). The fc-layers take up 96.1% of the overall storage space, as shown in Table 1. After the pruning, the network can be reduced to 10.1%, as shown in Table 2c. After the second assessment and third optimization step, DeepSZ uses 7×10^{-3}, 7×10^{-3}, and 5×10^{-3} as the error bound for fc6, fc7, and fc8, respectively, as shown in Figure 5d. DeepSZ can compress AlexNet by 45.5× with only 0.13% loss of top-1 accuracy, as shown in Table 3. Note that the top-5 accuracy is not decreased but, rather, is increased by 0.18%. We can further set the expected inference accuracy loss to zero. DeepSZ then can compress AlexNet by 36.5× with no loss of inference accuracy (the error bound of 2×10^{-3} for all fc6, fc7, and fc8).

VGG-16 on ImageNet. We now apply DeepSZ on VGG-16, which contains one large fc-layer (i.e., fc6) and two relatively small fc-layers (i.e., fc7 and fc8). The pruning ratios are set to relatively low values, leading to a much higher compression ratio after the pruning (i.e., 20.9×), as shown in Table 2d. DeepSZ then uses 10^{-2}, 9×10^{-3}, and 5×10^{-3} as the error bound for fc6, fc7, and fc8, respectively. By leveraging DeepSZ, we can achieve a compression ratio of 115.6× with only 0.25% loss of inference accuracy on VGG-16, as shown in Table 3. Similar to AlexNet, we can also set the expected inference accuracy loss to zero. DeepSZ then can compress VGG-16 by 92.7× with no loss of inference accuracy (with the error bound of 3×10^{-3} for fc6 and fc7 and 10^{-3} for fc8).

In summary, DeepSZ can compress the fc-layers in the tested neural networks with compression ratios of 57× to 116× while maintaining a loss of inference accuracy less than 0.3% (within the user-set expected loss of 0.4%), as shown in Table 3. Note that the top-5 accuracy is usually not displayed on the LeNet-5 because its top-1 accuracy (i.e., > 99%) is relatively high. DeepSZ can improve the overall compression ratio by 21% to 43%, compared with the second-best solution, as shown in Table 4. This table also illustrates

Table 3: Inference accuracy of DeepSZ compressed LeNet-5, AlexNet, and VGG-16.

Neural Network	Top-1 Accuracy	Top-5 Accuracy	fc-layers' Size	Compress Ratio
LeNet-300-100 original	98.35%	-	1056 KB	
LeNet-300-100 DeepSZ	98.31%	-	**19.1 KB**	55.8×
LeNet-5 original	99.13%	-	1620 KB	
LeNet-5 DeepSZ	99.16%	-	**28.3 KB**	57.3 ×
AlexNet original	57.41%	80.40%	234.5 MB	
AlexNet DeepSZ	57.28%	80.58%	**5.15 MB**	45.5 ×
VGG-16 original	68.05%	88.34%	494.5 MB	
VGG-16 DeepSZ	67.80%	88.20%	**4.277 MB**	115.6 ×

Table 4: Comparison of compression ratios of different techniques on LeNet-300-100, LeNet-5, AlexNet, and VGG-16.

Neural Network	Layer	Compression Ratio			
		Deep Compression	Weightless	DeepSZ	Improvement
LeNet-300-100	ip1	43.1	60.1	61.81	1.43×
	ip2	32.9	64.3	37.97	1.15×
	ip3	7.9	-	5.6	0.71×
	overall	**41.0**	**7.6**	**55.77**	**1.36×**
LeNet-5	ip1	40.8	74.2	58.5	1.43×
	ip2	16.3	-	21.5	1.32×
	overall	**40.1**	**39.0**	**57.3**	**1.43×**
AlexNet	fc6	41.8	-	54.4	1.30×
	fc7	40.7	-	46.5	1.14×
	fc8	17.1	-	17.5	1.02×
	overall	**37.7**	-	**45.5**	**1.21×**
VGG-16	fc6	119.0	157.0	152.1	1.28×
	fc7	80.0	85.8	90.0	1.13×
	fc8	19.1	-	19.8	1.04×
	overall	**95.8**	**5.9**	**115.6**	**1.21×**

that DeepSZ can deliver a high compression ratio for each fc-layer. Even compared with the Weightless method, which can compress only one layer, DeepSZ can still achieve a comparable compression ratio. We note that compression ratio is not available for some layers in Weightless, because (1) Weightless [33] does not provide their open source code and (2) the Weightless paper [33] showed evaluation results only for the largest two layers in LeNet-5 and VGG-16, without any results for AlexNet. We also note that Deep Compression uses 5 bits per pruned weights, whereas DeepSZ can compress the networks to 2.0 ∼ 3.3 bits per pruned weights. If we also set similar bit width for Deep Compression's quantization (i.e., the number of bits based on DeepSZ compressed layers), the inference accuracy will drop sharply by 1.56% for AlexNet and 2.81% for VGG-16, as shown in Table 5. Note that the inference accuracy degradation is not available for Weightless for LeNet-5 and AlexNet, because Weightless does not provide these results in [33] (the paper does show the inference accuracy degradation and encoding time overhead for VGG-16).

5.2.3 Performance Evaluation. As discussed in Section 4, DeepSZ is faster than the other methods theoretically in terms of both encoding and decoding. We now present the time overhead of DeepSZ on the four neural networks, as shown in Figure 7. The figure illustrates that DeepSZ has lower encoding and decoding time overheads than do Deep Compression and Weightless. We

[7]The accuracy is slightly increased by 0.03% in this case.

Table 5: Inference accuracy degradation of different techniques based on comparable compression ratio.

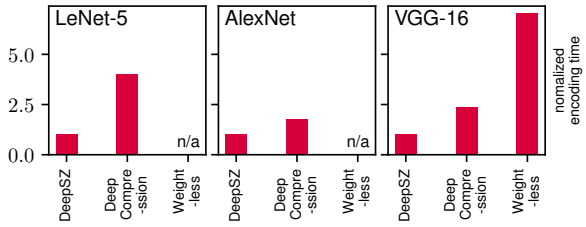
model	quantization (Deep Compression)	Bloomier Filter (Weightless)	SZ (DeepSZ)
LeNet-300-100	0.22%	-	0.12%
LeNet-5	0.30%	-	−0.03%[7]
AlexNet	1.56%	-	0.13%
VGG-16	2.81%	>3.0%	0.25%

note that the time results of LeNet-300-100 are almost identical to those of LeNet-5; hence, because of space limitations, we present the time overheads only for LeNet-5.

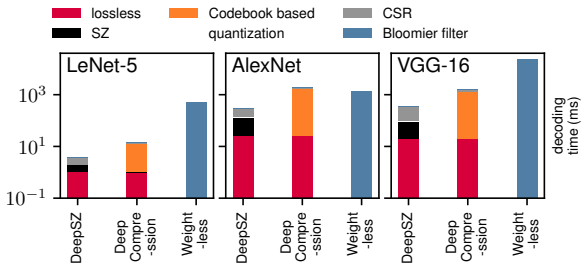
We investigated the times of the last three steps (i.e., spent mainly in the time of compression, decompression, and tests) for DeepSZ's encoding on GPUs. We do not include the pruning time because all three methods have the same pruning process and the time overheads are the same. Figure 7a shows the encoding time with the three solutions. We normalize the other two compression methods compared with DeepSZ in Figure 7a, because compared with AlexNet and VGG-16, LeNet-5 features much smaller encoding time. Specifically, DeepSZ takes <1 min, 8 min, and 16 min on encoding LeNet-5, AlexNet, and VGG-16, respectively. Deep Compression takes 4 min, 14 min, and 38 min on encoding LeNet-5, AlexNet, and VGG-16, respectively. Due to lack of source code, we estimate the encoding time of Weightless based on the number of epochs (for retraining) shown in the paper and the time of one epoch based on our experimental platform. Weightless takes about 113 min on encoding VGG-16; again, Weightless does not present the encoding time (i.e., the number of epochs) of LeNet-5 or AlexNet in the paper. DeepSZ can improve the encoding performance by 1.8× to 4.0× compared with the second-best solution. We note that for the Deep Compression and Weightless methods, it is difficult to determine the initial parameters of the solver in order to retrain the network. It could take much longer time than the optimal performance overhead if users are not familiar with the characteristics of the network.

We also investigated the times of lossless decompression, SZ lossy decompression, and sparse matrix reconstruction for DeepSZ's decoding on CPU. As we can see in Figure 7b, DeepSZ outperforms the second-best solution by 4.5× to 6.2× for decoding. Specifically, DeepSZ takes 2.7 ms, 296 ms, and 341 ms on decoding LeNet-5, AlexNet, and VGG-16, respectively; Deep Compression takes 13.9 ms, 1,832 ms, and 1,565 ms on decoding LeNet-5, AlexNet, and VGG-16, respectively; and Weightless takes 520 ms, 1,300 ms, and 22,800 ms on decoding LeNet-5, AlexNet, and VGG-16, respectively, as shown in the paper [8]. More specifically, for example, DeepSZ spends 26 ms in lossless decompression, 108 ms in SZ lossy decompression, and 162 ms in reconstructing the sparse matrix on AlexNet. As a comparison, the time for one forward pass with 50 images per batch takes 1,100 ms on AlexNet. This demonstrates that the time overhead of DeepSZ's decoding is comparatively low compared with typical forward pass. Therefore, once the network is needed for inference, DeepSZ can quickly decompress the compressed data and reconstruct the network without much delay. Note that the decoding time of Weightless relies on the number of non-pruned weights, whereas the decoding times of DeepSZ and Deep

[8]The paper evaluated its decoding time on an Intel Core i7-6700K Processor, which has similar processing power to our processor.



(a) Normalized encoding time with different solutions on GPUs.



(b) Breakdown of decoding time with different solutions on CPU.

Figure 7: Time breakdown of encoding and decoding with different lossy compression techniques.

Compression depend on the number of pruned weights. This difference can explain the following two observations. (1) DeepSZ and Deep Compression have similar decoding time on AlexNet and VGG-16 because they have similar numbers of pruned weights (i.e., 6.5 million for AlexNet and 5.8 million for VGG-16). (2) Weightless spends more time on VGG-16 than AlexNet for decoding because the largest `fc`-layer of VGG-16 (i.e., `fc6` of 25,088×4,096) is much larger than that of AlexNet (i.e., `fc6` of 9,216×4,096).

6 RELATED WORK

Most of neural networks have significant redundancy in their parameters according to a well-known research study [8]. Such redundant information may cause significant waste of computation, memory, and storage resources. In general, two types of methods have been proposed to resolve this issue: (1) modifying the network structures to reduce the complexity of parameters and (2) compressing a trained network by removing redundant information.

Modifying the structures of networks by adopting specialized structure or loss function can reduce the memory footprint while training the larger-scale networks with the same resources. For example, Vanhoucke et al. [44] exploited a fixed-point representation of activations with 8-bit integer rather than 32-bit floating point. Denton et al. [9] proposed using low-rank tensor approximations to reduce the number of parameters by up to a factor of 13 for a single layer while keeping the inference accuracy loss of 1% compared with the original network. Arora et al. [4] theoretically studied using random-like sparse networks with +1/0/-1 weights for interesting properties. Chen et al. [6] proposed a network architecture, named HashedNets, that uses a low-cost hash function to randomly group connection weights into hash buckets, such that all connections within the same hash bucket share a single value.

Compressing neural networks is an alternative strategy to reduce the model size. For example, Gong et al. [13] compressed `fc`-layers by using vector quantization, which achieved a compression ratio of 24 with 1% inference accuracy loss. Recently, two state-of-the-art works [15, 33] have been designed for compressing the network with high compression ratio and inference accuracy. Han et al. proposed a three-step approach, named Deep Compression, that contains pruning, quantization, and encoding. Deep Compression, however, may degrade the inference accuracy significantly in the course of each forward-propagation because of its vector quantization design, such that the network has to to be retrained over and over again in order to reach the target inference accuracy, thus resulting in a high execution time overhead. Reagen et al. proposed a lossy compression method, named Weightless, by adopting a Bloomier filter to compress the weights lossily. For encoding, the Bloomier filter needs to construct a hash table, which is $O(n \log n)$ in time complexity; for decoding, in order to decompress one value, the Bloomier filter typically needs to calculate four hashing functions. The time complexity is $O(n)$ for the best case but $O(n^2)$ for the worst case. Here n is the number of values for encoding/decoding. Therefore, the Weightless method suffers from a relatively high time overhead because of the expensive Bloomier filter. Moreover, it was applied to only one `fc`-layer instead of the whole neural network. Tung et al. [43] proposed a method named CLIP-Q that uses weight pruning and quantization, which is similar to Deep Compression. Unlike Deep Compression that separates pruning and quantization, CLIP-Q combines them at the same step in a single framework and can be performed in parallel with network fine-tuning. Moreover, it adopts much higher pruning ratio than Deep Compression in order to achieve higher overall compression ratio. In this paper, we mainly compare our proposed DeepSZ with both Deep Compression and Weightless approaches comprehensively.

Unlike the first type of method that requires modification of the network structure and full retraining, the second type of method is more general and efficient. Therefore, we focus on compressing well-trained neural networks without modifying the network structure for high reduction ratio and inference accuracy.

7 CONCLUSION AND FUTURE WORK

In this paper, we propose a novel lossy compression framework, called DeepSZ, for effectively compressing sparse weights in deep neural networks. Unlike traditional methods, DeepSZ can avoid the costly retraining process after compression, leading to a significant performance improvement in encoding DNNs. We develop a series of approaches to efficiently determine the best-fit error bound for each layer in the network, maximizing the overall compression ratio with user acceptable loss of inference accuracy. Experimental results based on the tested neural networks show that DeepSZ can achieve compression ratios of up to 116× and can outperform the second-best approach by up to 1.43×. Our experiments with four Nvidia Tesla V100 GPUs demonstrate that DeepSZ can obtain 1.8× to 4.0× performance improvement in encoding compared with the previous state-of-the-art. DeepSZ can improve the decoding performance by 4.5× to 6.2× compared with the second-best solution. DeepSZ also can provide high flexibility to balance the compression ratio and inference accuracy. We plan to first evaluate our proposed DeepSZ on more neural network architectures. We also will further improve the SZ compression algorithm to achieve a higher reduction ratio in compressing DNNs. Moreover, we hope to use DeepSZ for improving GPU memory utilization.

ACKNOWLEDGMENTS

This research was supported by the Exascale Computing Project (ECP), Project Number: 17-SC-20-SC, a collaborative effort of two DOE organizations – the Office of Science and the National Nuclear Security Administration, responsible for the planning and preparation of a capable exascale ecosystem, including software, applications, hardware, advanced system engineering and early testbed platforms, to support the nation's exascale computing imperative. This material was based upon work supported by the U.S. Department of Energy, Office of Science, under contract DE-AC02-06CH11357, and also supported by the National Science Foundation under Grant No. 1619253. We gratefully acknowledge the support from Alabama Water Institute (AWI), Remote Sensing Center (RSC), and Center for Complex Hydrosystems Research (CCHR).

REFERENCES

[1] Mark Ainsworth, Ozan Tugluk, Ben Whitney, and Scott Klasky. 2018. Multilevel techniques for compression and reduction of scientific dataâĂŤthe univariate case. *Computing and Visualization in Science* 19, 5–6 (2018), 65–76.
[2] F Alted. 2017. Blosc, an extremely fast, multi-threaded, meta-compressor library.
[3] Manoj Alwani, Han Chen, Michael Ferdman, and Peter Milder. 2016. Fused-layer CNN accelerators. In *The 49th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Press, 22.
[4] Sanjeev Arora, Aditya Bhaskara, Rong Ge, and Tengyu Ma. 2014. Provable bounds for learning some deep representations. In *International Conference on Machine Learning*. 584–592.
[5] Blosc compressor. 2018. http://blosc.org/. Online.
[6] Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. 2015. Compressing neural networks with the hashing trick. In *International Conference on Machine Learning*. 2285–2294.
[7] Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning*. ACM, 160–167.
[8] Misha Denil, Babak Shakibi, Laurent Dinh, Nando De Freitas, et al. 2013. Predicting parameters in deep learning. In *Advances in neural information processing systems*. 2148–2156.
[9] Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. 2014. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in neural information processing systems*. 1269–1277.
[10] Peter Deutsch. 1996. *GZIP file format specification version 4.3.* Technical Report.
[11] Sheng Di and Franck Cappello. 2016. Fast error-bounded lossy HPC data compression with SZ. In *2016 IEEE International Parallel and Distributed Processing Symposium*. IEEE, 730–739.
[12] Denis Foley and John Danskin. 2017. Ultra-performance Pascal GPU and NVLink interconnect. *IEEE Micro* 37, 2 (2017), 7–17.
[13] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. 2014. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115* (2014).
[14] GSMA. 2014. Half of the worlds population connected to the mobile internet by 2020. https://www.gsma.com/newsroom/press-release/half-worlds-population-connected-mobile-internet-2020-according-gsma/.
[15] Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* (2015).
[16] Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*. 1135–1143.
[17] Babak Hassibi and David G Stork. 1993. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in neural information processing systems*. 164–171.
[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 770–778.
[19] Lawrence Ibarria, Peter Lindstrom, Jarek Rossignac, and Andrzej Szymczak. 2003. Out-of-core compression and decompression of large n-dimensional scalar fields. In *Computer Graphics Forum*, Vol. 22. Wiley Online Library, 343–348.
[20] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross B Girshick, Sergio Guadarrama, and Trevor Darrell. 2014. Caffe - Convolutional Architecture for Fast Feature Embedding. *ACM Multimedia* (2014).
[21] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.
[22] Sriram Lakshminarasimhan, Neil Shah, Stephane Ethier, Scott Klasky, Rob Latham, Rob Ross, and Nagiza F Samatova. 2011. Compressing the incompressible with ISABELA: In-situ reduction of spatio-temporal data. In *European Conference on Parallel Processing*. Springer, 366–379.
[23] Large Scale Visual Recognition Challenge. 2019. http://www.image-net.org/challenges/LSVRC/. Online.
[24] Yann LeCun et al. 2015. LeNet-5, convolutional neural networks. *URL: http://yann.lecun.com/exdb/lenet* 20 (2015).
[25] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.
[26] Yann LeCun, Corinna Cortes, and CJ Burges. 2010. MNIST handwritten digit database. *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist* 2 (2010).
[27] Yann LeCun, John S Denker, and Sara A Solla. 1990. Optimal brain damage. In *Advances in neural information processing systems*. 598–605.
[28] Xin Liang, Sheng Di, Dingwen Tao, Sihuan Li, Shaomeng Li, Hanqi Guo, Zizhong Chen, and Franck Cappello. 2018. Error-Controlled Lossy Compression Optimized for High Compression Ratios of Scientific Datasets. (2018).
[29] Peter Lindstrom. 2014. Fixed-rate compressed floating-point arrays. *IEEE Transactions on Visualization and Computer Graphics* 20, 12 (2014), 2674–2683.
[30] Peter Lindstrom. 2017. Error Distributions of Lossy Floating-Point Compressors. *Joint Statistical Meetings* (2017), 2574–2589.
[31] Peter Lindstrom and Martin Isenburg. 2006. Fast and efficient compression of floating-point data. *IEEE Transactions on Visualization and Computer Graphics* 12, 5 (2006), 1245–1250.
[32] Tao Lu, Qing Liu, Xubin He, Huizhang Luo, Eric Suchyta, Jong Choi, Norbert Podhorszki, Scott Klasky, Mathew Wolf, Tong Liu, et al. 2018. Understanding and modeling lossy compression schemes on HPC scientific data. In *2018 IEEE International Parallel and Distributed Processing Symposium*. IEEE, 348–357.
[33] Brandon Reagan, Udit Gupta, Bob Adolf, Michael Mitzenmacher, Alexander Rush, Gu-Yeon Wei, and David Brooks. 2018. Weightless: Lossy weight encoding for deep neural network compression. In *International Conference on Machine Learning*. 4321–4330.
[34] Henry A Rowley, Shumeet Baluja, and Takeo Kanade. 1998. Neural network-based face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20, 1 (1998), 23–38.
[35] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).
[36] Seung Woo Son, Zhengzhang Chen, William Hendrix, Ankit Agrawal, Wei-keng Liao, and Alok Choudhary. 2014. Data compression for the exascale computing era-survey. *Supercomputing Frontiers and Innovations* 1, 2 (2014), 76–88.
[37] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 1–9.
[38] Dingwen Tao, Sheng Di, Zizhong Chen, and Franck Cappello. 2017. In-depth exploration of single-snapshot lossy compression techniques for N-body simulations. In *2017 IEEE International Conference on Big Data*. IEEE, 486–493.
[39] Dingwen Tao, Sheng Di, Zizhong Chen, and Franck Cappello. 2017. Significantly improving lossy compression for scientific data sets based on multidimensional prediction and error-controlled quantization. In *2017 IEEE International Parallel and Distributed Processing Symposium*. IEEE, 1129–1139.
[40] Dingwen Tao, Sheng Di, Xin Liang, Zizhong Chen, and Franck Cappello. 2019. Optimizing Lossy Compression Rate-Distortion from Automatic Online Selection between SZ and ZFP. *IEEE Transactions on Parallel and Distributed Systems* (2019).
[41] David Taubman and Michael Marcellin. 2012. *JPEG2000 image compression fundamentals, standards and practice: image compression fundamentals, standards and practice*. Vol. 642. Springer Science & Business Media.
[42] Surat Teerapittayanon, Bradley McDanel, and HT Kung. 2017. Distributed deep neural networks over the cloud, the edge and end devices. In *2017 IEEE 37th International Conference on Distributed Computing Systems*. IEEE, 328–339.
[43] Frederick Tung and Greg Mori. 2018. CLIP-Q: Deep network compression learning by in-parallel pruning-quantization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 7873–7882.
[44] Vincent Vanhoucke, Andrew Senior, and Mark Z Mao. 2011. Improving the speed of neural networks on CPUs. In *Proc. Deep Learning and Unsupervised Feature Learning NIPS Workshop*, Vol. 1. Citeseer, 4.
[45] Hao Wang, Naiyan Wang, and Dit-Yan Yeung. 2015. Collaborative deep learning for recommender systems. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*. ACM, 1235–1244.
[46] Linnan Wang, Jinmian Ye, Yiyang Zhao, Wei Wu, Ang Li, Shuaiwen Leon Song, Zenglin Xu, and Tim Kraska. 2018. Superneurons: dynamic GPU memory management for training deep neural networks. In *Proceedings of the 23rd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. ACM, 41–53.
[47] You-Chiun Wang, Yao-Yu Hsieh, and Yu-Chee Tseng. 2008. Compression and storage schemes in a sensor network with spatial and temporal coding techniques. In *VTC Spring 2008-IEEE Vehicular Technology Conference*. IEEE, 148–152.
[48] Yi Zheng, Qi Liu, Enhong Chen, Yong Ge, and J Leon Zhao. 2014. Time series classification using multi-channels deep convolutional neural networks. In *International Conference on Web-Age Information Management*. Springer, 298–310.
[49] Zstandard. 2018. http://facebook.github.io/zstd/. Online.