Delta-DNN: Efficiently Compressing Deep Neural Networks via Exploiting Floats Similarity

Zhenbo Hu[§,*], Xiangyu Zou[§,*], Wen Xia[§,$], Sian Jin[ζ], Dingwen Tao[ζ], Yang Liu[§,$], Weizhe Zhang[§,$], Zheng Zhang[§]

[§]School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, China
[$]Cyberspace Security Research Center, Peng Cheng Laborary, Shenzhen, China
[ζ]School of Electrical Engineering and Computer Science, Washington State University, WA, USA
zhenbohu.hitsz@gmail.com,xiangyu.zou@hotmail.com
{xiawen,liu.yang,wzzhang,zhengzhang}@hit.edu.cn
{sian.jin,dingwen.tao}@wsu.edu

ABSTRACT

Deep neural networks (DNNs) have gained considerable attention in various real-world applications due to the strong performance on representation learning. However, a DNN needs to be trained many epochs for pursuing a higher inference accuracy, which requires storing sequential versions of DNNs and releasing the updated versions to users. As a result, large amounts of storage and network resources are required, significantly hampering DNN utilization on resource-constrained platforms (e.g., IoT, mobile phone).

In this paper, we present a novel delta compression framework called Delta-DNN, which can efficiently compress the float-point numbers in DNNs by exploiting the floats similarity existing in DNNs during training. Specifically, (1) we observe the high similarity of float-point numbers between the neighboring versions of a neural network in training; (2) inspired by delta compression technique, we only record the delta (i.e., the differences) between two neighboring versions, instead of storing the full new version for DNNs; (3) we use the error-bounded lossy compression to compress the delta data for a high compression ratio, where the error bound is strictly assessed by an acceptable loss of DNNs' inference accuracy; (4) we evaluate Delta-DNN's performance on two scenarios, including reducing the transmission of releasing DNNs over the network and saving the storage space occupied by multiple versions of DNNs.

According to experimental results on six popular DNNs, Delta-DNN achieves the compression ratio 2×-10× higher than state-of-the-art methods, while without sacrificing inference accuracy and changing the neural network structure.

* Z. Hu and X. Zou equally contributed to this work.

ICPP '20, August 17–20, 2020, Edmonton, AB, Canada
© 2020 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-8816-0/20/08...$15.00
https://doi.org/10.1145/3404397.3404408

KEYWORDS

Lossy compression, neural network, delta compression

1 INTRODUCTION

In recent years, deep neural networks (DNNs) have been widely applied to many artificial intelligence tasks in various scientific and technical fields, such as computer vision [44], natural language processing [4]. Generally, DNNs are designed to solve complicated and non-linear problems (e.g., the face recognition [45]) by using multi-layer structures, which consist of millions of parameters (i.e., floating-point data types).

To further improve the data analysis capabilities by increasing the inference accuracy, DNNs are becoming deeper and more complicated [43]. Meanwhile, the frequent training of DNNs results in huge overheads for storage and network. The overheads overwhelm resource-constrained platforms such as mobile devices and IoT devices. For instance, a typical use case is to train a DNN in the cloud servers using high-performance accelerators, such as GPUs or TPUs, and then transfer the trained DNN model to the edge devices to provide accurate, intelligent, and effective services [20], while the cloud to edge data transfer is very costly.

Compressing neural networks [32] is an effective way to reduce the data transfer cost. However, existing approaches focus on simplifying DNNs for compression such as pruning [21, 33], low-rank approximation [14], quantization [12, 37], knowledge distillation [11], and compact network design [38, 55]. These methods usually operate on models that have already been trained, modifying the model structure and may make the model untrainable [30]. Therefore, in this paper, we focus on designing an approach to effectively compressing DNNs without changing their structures.

Existing studies [15, 24] suggest that DNNs are difficult to be compressed using traditional compressors (e.g., GZIP [5], LZMA [35]) since there are large amounts of floating-point numbers with the random ending mantissa bits. However, our observation on existing DNNs suggests that most of the floating-point values only slightly change during the training of DNNs. Thus, there exists data similarity between the neighboring neural networks[1] in training, which means we only need to record the delta (i.e., the differences) of

[1]Neighboring neural networks in this paper refers to the checkpointed/saved neural network models in the training epochs/iterations.

the data between two neighboring versions for potentially higher data reduction. Moreover, considering the locality of the delta, error-bounded lossy compression mechanisms can be very effective.

In this paper, we present a novel delta compression framework called Delta-DNN to efficiently compress DNNs with a high compression ratio by exploiting the data similarity existing in DNNs during training. Our contributions are four-fold:

(1) We observe the high similarity of floating-point numbers (i.e., parameters) existing in the neighboring versions of a DNN, which can be exploited for data reduction.
(2) We propose to only store the delta data between neighboring versions, then apply error-bounded lossy compression to the delta data for a high compression ratio. The error bound is strictly assessed by the maximal tolerable loss of DNNs' inference accuracy. To the best of our knowledge, this is the first work to employ the idea delta compression for efficiently reducing the size of DNNs.
(3) We discuss Delta-DNN's benefits on two typical scenarios, including significantly reducing the bandwidth consumption when releasing DNNs over the network and reducing the storage requirements by preserving versions of DNNs.
(4) Our evaluation on six popular DNNs suggests that compared with state-of-the-art compressors including SZ, LZMA, and Zstd, Delta-DNN can improve the compression ratio by $2\times$ to $10\times$ while keeping the neural network structure unchanged at the cost of only losing inference accuracy by less than 0.2%.

The rest of the paper is organized as follows. Section 2 presents the background and motivation of this work. Section 3 describes the design methodologies of the Delta-DNN framework in detail. Section 4 discusses the two typical application scenarios of Delta-DNN. Section 5 discusses the evaluation results of Delta-DNN on six well-known DNNs, compared with state-of-the-art compressors. In Section 6, we conclude this paper and present our future work.

2 BACKGROUND AND MOTIVATION

In this section, we present the necessary background about data compression techniques, compressibilities of DNNs, lossy compression of floats, and also discuss our critical observations of the data similarity existing in DNNs as our research motivation.

2.1 Data Compression Techniques

Recently, data grows exponentially in cloud computing, big data, and artificial intelligence environments. As a result, efficient data compression techniques are especially important for data reduction. Generally, data compression can be categorized into lossless compression and lossy compression techniques.

General lossless compression, such as GZIP [5], LZMA [35], and Zstd [57], has been developed for decades. These techniques usually deal with data as byte streams, and reduce data at the byte/string level based on classic algorithms such as Huffman coding [13] and dictionary coding [56]. These fine-grained data compression techniques are extremely compute-intensive, and are usually used to eliminate redundancies inside a file or in a limited data range.

Recently, some other **special lossless compression** techniques have been proposed, such as data deduplication [8, 31], and delta compression [42], which eliminates redundancies among files. For example, data deduplication, a coarse-grained compression technique, detects and eliminates the duplicate chunks/files among the different backup versions in backup storage systems. ***Delta compression*** *observes the high data similarity and thus the high data redundancies in the neighboring versions of the (modified) files in backup storage systems, and then records the delta data (the differences) of them for space savings [42, 51]*, by eliminating redundancies between the similar files at the byte- and string-level.

Lossy compression also appears for decades. The typical lossy compressors are for images, and their designs are based on human perception, such as JPEG2000 [36]. Thus they use lossy compression techniques, such as wavelet transform and vector optimization, to dramatically reduce the image data sizes.

Nowadays, more attention is paid to the lossy compression of floating-point data generated from HPC. The most popular lossy compressor includes ZFP [27], SZ [6, 26, 47]. ZFP is an error-controlled lossy compressor designed for scientific data. Specifically, ZFP first transforms the floating-point numbers to fixed-point data values block by block. Then, a reversible orthogonal block transformation is applied in each block to mitigate the spatial correlation, and embedded coding [39] is used to encode the coefficients.

SZ, another lossy compressor, is designed for scientific data focusing on a high compression ratio by fully exploiting the characteristics of floating-point numbers. Specifically, there is a data-fitting predictor in SZ's workflow, which generates a predicted value for each data according to its surrounding data (i.e., data's spatial correlation). Predictor utilizes the characteristics (or the rules) in floats for prediction while ensuring the point-wise error controls on demand. Consequently, the difference between the predicted value and the real value will be quantized, encoded, and batch compressed, which often achieve a much higher compression ratio than directly compressing the real value. *Note that SZ's compression ratio depends on the predictor design, and also the rules are existing in the data.* Thus, to achieve a high compression ratio, some variants of SZ try to explore more rules existing in the data, such as the temporal correlation and spatiotemporal decimation [23, 25].

2.2 Compressing DNNs

Recently, with the rise of AI boom, efficiently compressing deep neural networks (DNNs) are also gaining increasing attention. However, this is not easy, due to that DNNs consist of many floating-point numbers generated from training on a large amount of users' data, which is of very low compressibility.

A widely adopted method of training DNNs is called *Gradient Descent* [19], which usually generates large amounts of floating-point numbers and requires lots of resources (e.g., memory, storage, and network). More specifically, in the training process of a DNN, a snapshot will be generated periodically for backup, to prevent the case of accuracy decreasing by over-fitting the DNN. Thus, storing many versions of DNNs is space-consuming, and also the case of frequently dispatching a new version of DNN to users, is becoming a challenge for using DNN in practical, especially in the resource-limited scenarios, such as mobile phones and wireless sensors.

Generally, compressing DNNs means compressing a large amount of very random floating-point numbers. Due to the strong randomness of the ending mantissa bits in DNNs' floats, existing traditional compression approaches only achieve a very limited compression ratio on DNNs according to recent studies [28, 41], both lossless compressors (such as GZIP [5], Zstd [57], and LZMA [35]) and lossy compressors (such as SZ [6, 26, 47] and ZFP [27]).

Therefore, other special technologies for compressing DNNs are proposed, such as pruning [7, 21, 33] and quantization [12, 37]. Deep Compression [9] uses the techniques of pruning (removing some unimportant parameters), quantization (transforming the floats into integers), and Huffman coding to compress DNNs, which may change the neural network structure (by pruning) and may significantly degrade the networks' inference accuracy (by quantization and pruning). Therefore, the network needs to be retrained many times to avoid accuracy degradation, thus resulting in huge execution overheads. DeepSZ [15] combines SZ lossy compressor and the pruning technique to achieve a high compression ratio on DNNs, but it may also change the structure of networks due to pruning.

There are also some other approaches focusing on not only compressing the storage space of DNNs but also simplifying the models in DNNs to reduce computation, which is called structured pruning [21, 50]. For example, Wen et al. [50] added a LASSO (Least Absolute Shrinkage and Selection Operator) regular penalty term into the loss function to realize the structured sparsity of DNNs. Li et al. [21] proposed a method of pruning the filter of convolutional layers to remove the unimportant filters accordingly.

Overall, the pruning or quantization based approaches will destroy the completeness of the structure of DNNs, and may degrade the inference accuracy. In this paper, we focus on using the traditional compression approaches to compress the parameters of DNNs without changing DNNs' structures.

2.3 Observation and Motivation

As introduced in the last subsection, there are many parameters (i.e., floating-point numbers) in a neural network, and in the training process (i.e., *Gradient Descent*), some parameters in the networks vary greatly while some parameters vary slightly. According to our experimental observations, most of the parameters vary slightly during training, which means data similarity exists (i.e., data redundancy) among the neighboring neural network versions.

To study the similarity in DNNs, we collect 368 versions of six popular neural networks in training (six DNNs' characteristics and the training dataset will be detailed in Subsection5.1) and then linearly fit all the corresponding parameters from every two neighboring networks (including all layers) as shown in Figure 1. Besides, in this test, we use the metric of Structural Similarity (SSIM) [49] (also shown in Figure 1), which is widely used to measure the similarity of two bitmap pictures. Because bitmap pictures and neural networks both can be regarded as matrices, it is reasonable to use this metric to measure the similarity of two networks.

Observation: the results shown in Figure 1 suggest the floating-point numbers of the neighboring networks are very similar: ① the results of the linear fitting are very close to $y = x$. Specifically, in Figure 1, the x-axis and y-axis denote the parameters of two neighboring network versions, respectively. We can observe a clear linear relationship, which reflects the extremely high similarity between the neighboring neural network models. ② the SSIM is very close to 1.0, which usually suggests that bitmaps are very similar [49].

Although there exists data similarity, parameters in neighboring versions of DNNs are completely different in terms of bit representation due to the random ending mantissa bits. Thus, traditional lossless compression methods can not achieve an ideal compression ratio on these similar floats in DNNs. Meanwhile, SZ compressor can well compress the similar floats (i.e., the rule existing in floats) by using a data-fitting predictor and an error-controlled quantizator.

Therefore, according to the above observation and discussion, we present the motivations of this paper: **Motivation ①**, inspired by the delta compression technique used in the area of backup storage, we can calculate the delta data (i.e., difference) of the similar floats between neighboring networks, which is very compressible in the lossy compression; **Motivation ②**, we employ the ideas of error-bound SZ lossy compression, i.e., a data-fitting predictor and an error-controlled quantizator, to compress the delta data while ensuring the inference accuracy loss under control.

The two motivations, combined with our observations of floats similarity existing in DNNs, inspire us to adopt delta compression and error-bounded lossy compression with high compressibility. Specifically, regarding there exists floats similarity between two neighboring versions of a DNN (see Figure 1), we propose an "inter-version" predictor to predict the corresponding data value between two neighboring network versions, which will achieve a much higher compression ratio (as demonstrated in Section 5), and the specific techniques will be introduced in Section 3.

Note that DeepSZ [15] also uses the error-bounded SZ lossy compression to compress DNNs, and the key difference between DeepSZ and our approach Delta-DNN is that DeepSZ directly compresses floats one-by-one on DNNs combining with some pruning techniques (may change the network structure [30]). At the same time, Delta-DNN exploits the floats similarity of neighboring networks for lossy compression to target at a much higher compression ratio on DNNs regardless of whether or not to use the pruning techniques.

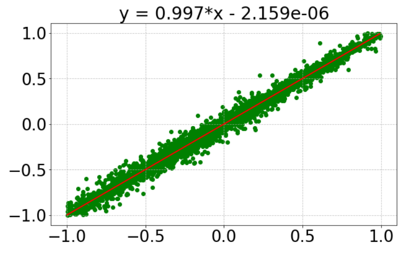
3 DESIGN AND IMPLEMENTATION

Generally, Delta-DNN runs after the time-consuming training or fine-tuning of the network on the training datasets (to get the updated neural networks). In this section, we describe Delta-DNN design in detail, including exploiting similarity of the neighboring neural networks to calculate the lossy delta data, optimizing parameters, and encoding schemes.

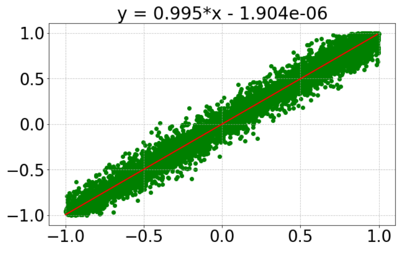
3.1 Overview of Delta-DNN Framework

The general workflow of Delta-DNN framework is shown in Figure 2. To compress a neural network (called target network), we need a reference neural network, which is usually the former version of the network in training, and Delta-DNN will calculate and compress the delta data of two networks for efficient space savings. Specifically, Delta-DNN consists of three key steps: calculating the delta data, optimizing the error bound, and compressing the delta data.

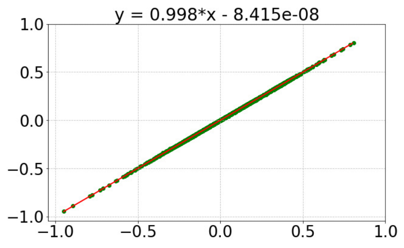
(1) *Calculating the delta data* is to calculate the lossy delta data of the target and reference networks (including **all layers**), which will be much more compressible than directly compressing the floats in DNNs, as detailed in Subsection 3.2.
(2) *Optimizing the error bound* is to select the suitable error bound used for maximizing the lossy compression efficiency



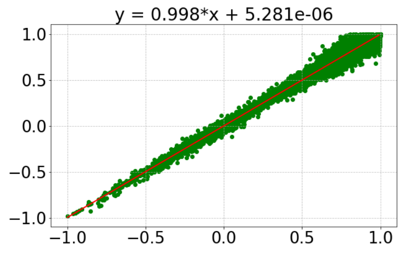
(a) VGG-16, SSIM: 0.99994



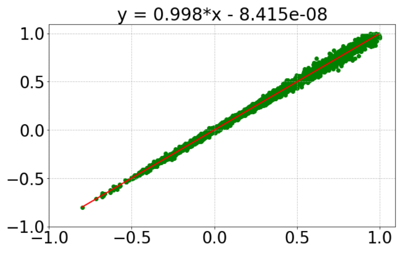
(b) ResNet101, SSIM: 0.99971



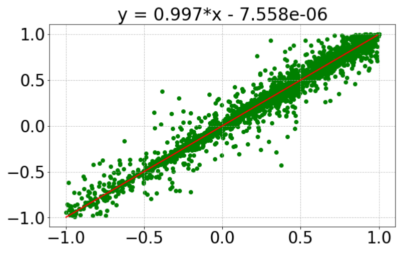
(c) GoogLeNet, SSIM: 0.99999



(d) EfficientNet, SSIM: 0.99624



(e) MobileNet, SSIM: 0.99998



(f) ShuffleNet, SSIM: 0.99759

Figure 1: Parameter similarity (using linear fitting) of the neighboring neural networks in training, from the six popular DNNs.

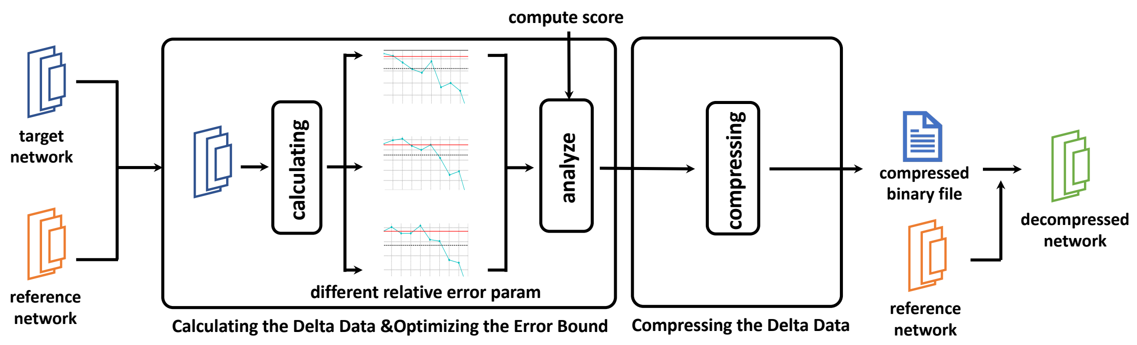


Figure 2: Overview of Delta-DNN framework for compressing deep neural networks.

while meeting the requirements of the maximal tolerable loss of DNNs' inference accuracy, as detailed in Subsection 3.3.

(3) *Compressing the delta data* is to reduce the delta data size by using the lossless compressors, as detailed in Subsection 3.4.

In the remainder of this section, we will discuss these three steps in detail to show how Delta-DNN efficiently compresses the floating-point numbers by exploiting their similarity.

3.2 Calculating the Delta Data

In Delta-DNN, because of the observed floats similarity existing in the neighboring networks, the corresponding parameters (i.e., floats) of the target network and reference network will be calculated their lossy delta data, following the idea of SZ lossy compressor [6, 26, 47]. Specifically, we denote a parameter from the target network as A_i and the corresponding parameters from the reference network as B_i, and the lossy delta data of the floats A_i and B_i can be calculated and quantized as below.

$$M_i = \lfloor \frac{A_i - B_i}{2 \cdot log(1+\epsilon)} + 0.5 \rfloor \qquad (1)$$

Here ϵ is the predefined relative error bound used for SZ lossy compression (e.g., 1E-1 or 1E-2), and M_i is an integer (called 'quantization factor') for recording the delta data of A_i and B_i. This delta calculation of floats shown in Equation (1) is demonstrated to be very efficient in lossy compression of scientific data (i.e., compressing large amounts of floats) where the 'quantization factors' (i.e., the integers M_i) are highly compressible [26].

In Delta-DNN, according to the similarity we observed in Subsection 2.3, we believe that $A_i - B_i$ will be very small in most cases, which means most of M_i are equal to zero. Therefore, the 'quantization factors' M are also very compressible in Delta-DNN as that in Scientific data. Then the target network will be replaced by "the M_i network" (the lossy delta data) for space savings in Delta-DNN.

With the delta data (i.e., amounts of M_i) and the reference network (i.e., B_i), we can recover (i.e., decompress) the target network (i.e., the parameters A_i) as illustrated in Equation (2) with a limited loss based on the error bound ϵ: $|A_i - A_i'| < \epsilon$.

$$A_i' = 2 \cdot M_i \cdot log(1+\epsilon) + B_i \tag{2}$$

All in all, the delta data M_i between the target and reference networks are very compressible, while its compression ratio depends on the two key factors. The first factor is the similarity of the data A_i and B_i, which is demonstrated to be very high in Subsection 2.3. The second one is the predefined relative error bound ϵ used for lossy compression, which also impacts the inference accuracy of DNNs in our lossy delta compression framework. The selection of the error bound ϵ used in our Delta-DNN will be discussed in Subsection 3.3.

3.3 Optimizing the Error Bound

In this subsection, we discuss how to get a reasonable relative error bound ϵ to maximize the compression ratio of Delta-DNN without compromising DNNs' inference accuracy.

From Equation (1) described in Subsection 3.2, we can know that the larger error bound results in the smaller 'quantization factor' and thus the higher compression ratio. Nevertheless, at the same time, it leads to an uncertain inference accuracy loss of DNNs. This is because the recovered target network parameters would vary randomly (i.e., sometimes larger, sometimes smaller) along with different error bounds, after decompression in Delta-DNN.

Figures 3 and 4 show examples of studying the impact of the error bound ϵ on the final compression ratio on DNNs, with Delta-DNN running on six well known DNNs. Generally, the results demonstrate our discussion in the last paragraph that Delta-DNN can achieve a higher compression ratio (see Figure 4) but cause uncertain inference accuracy when increasing the error bounds (see Figure 3).

Note that Figure 3 shows of the inference accuracy of the last model files on six DNNs using Delta-DNN with different error bounds. Figure 3 (c) has a significant accuracy decrease, and we also observe this phenomenon among the different training stages in other DNNs. This is because gradient descents are nonlinear, and some steps of them seem to be more sensitive.

Therefore, in Delta-DNN's workflow of calculating the delta data, we need to find a reasonable error bound for both considering the two key metrics: *the compression ratio* and *the inference accuracy loss* on DNNs. Moreover, we need to design a flexible solution to meet the users' various requirements on both the two metrics while strictly ensuring the inference accuracy loss is acceptable for DNNs (e.g., 0.2% as shown in Figure 3).

To this end, Algorithm 1 presents our method of selecting an optimal error bound by assessing its impact on both compression ratio and inference accuracy loss on DNNs. Generally, it consists of two steps: ① Collecting the results of compression ratio and the inference accuracy degradation along with the available error bounds, while meeting the requirement of the maximal tolerable accuracy loss of the tested DNN, ② Assessing the collected results to select an optimal error bound according to Formula (3) as below.

$$Score = \alpha \cdot \Phi + \beta \cdot \Omega, \ (\alpha + \beta = 1) \tag{3}$$

where Φ is the relative loss of inference accuracy (calculated the recovered network after decompression, compared with the original network); Ω is the compression ratio (i.e., the ratio of before/after compression); α and β are the influencing weights defined by users, which are used to fine-tune the importance of Φ and Ω in Formula

Algorithm 1: Error Bound Assessment and Selection

```
Input: Target network: $N_1$;   Reference network: $N_2$;
Accepted accuracy loss: $\theta$;   Available error bounds: $EB$;
Compression ratio of network $N$ with error bound $\epsilon$: $\Omega(N,\epsilon)$;
Accuracy degradation of network $N$ with error bound $\epsilon$: $\Phi(N,\epsilon)$;
Output: The best error bound: $EB_{best}$;
//$\alpha, \beta$ are weights of compression ratio & accuracy defined by user;
for $\epsilon$ in $EB$ do
    $\{\Phi(N_1,\epsilon), \Omega(N_1,\epsilon)\} \leftarrow Estimate(N_1, N_2, \epsilon)$;
    if $abs(\Phi(N_1,\epsilon)) < \theta$ then
        save $\{\Phi(N_1,\epsilon), \Omega(N_1,\epsilon)\}$ in $Sets$;
$SCORE_{best} \leftarrow 0$;
$EB_{best} \leftarrow \lambda$;   //$\lambda$ is user-defined default minimal error bound;
for $\{\Phi(N_1,\epsilon), \Omega(N_1,\epsilon)\}$ in $Sets$ do
    $Score \leftarrow CalcScore(\Phi(N_1,\epsilon), \Omega(N_1,\epsilon), \alpha, \beta)$;
    if $Score > SCORE_{best}$ then
        $SCORE_{best} \leftarrow Score$;
        $EB_{best} \leftarrow \epsilon$;
return $EB_{best}$;
```

(3), and $\alpha + \beta = 1$ (e.g., the user can set $\alpha = 0, \beta = 1$ to maximize the importance of compression ratio in Delta-DNN).

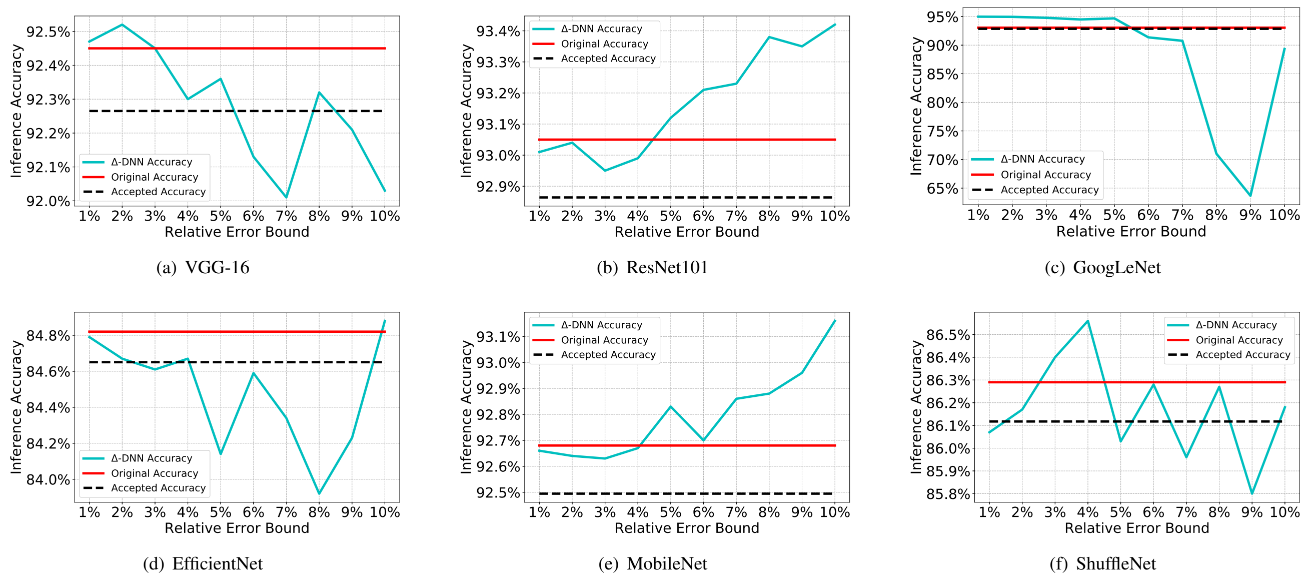
Thus, as shown in Formula (3) and Algorithm 1, we use the *Score* to assess the compression efficiency of Delta-DNN to select the optimal error bound for DNNs, satisfying users' requirements on both the compression ratio and the inference accuracy of DNNs.

Note that the computation cost of Algorithm 1 is minor compared with the training process of DNNs, which is explained as below. Generally, the time complexity of Algorithm 1 is $O(n \cdot (\tau + e + d))$, where n is the number of error bounds for testing, and n is equal to 10 in this paper; $O(\tau)$ is the time complexity of testing a network's inference accuracy on a testing dataset; $O(e)$ is the time complexity of compressing a network, and $O(d)$ is the time complexity of decompressing a network (usually negligible [15]). The time costs of compressing and testing (for DNN accuracy) are both positively related to the size of the network while compressing is usually faster than testing (for DNN accuracy). Hence, the time complexity of Algorithm 1 can be simplified to $O(n \cdot k \cdot \tau)$ where $1 < k < 2$. Delta-DNN is running after the time-consuming training or fine-tuning of the network on the training datasets. Specifically, in deep neural networks, the time complexity of training DNN on the training datasets $O(M)$ over that of verifying DNN's accuracy on the testing datasets, is usually 99:1 [2], so the time complexity of Algorithm 1 is about $O(\frac{n \cdot k \cdot M}{99}) \approx O(\frac{M}{5})$, which is much smaller than the training time complexity $O(M)$ in deep learning.

3.4 Compressing the Delta Data

After calculating the lossy delta data (i.e., 'quantization factor' as introduced in Equation (1) in Subsection 3.2) with the optimized error bound according to the requirement of the inference accuracy of DNNs, Delta-DNN then compresses these delta data using the lossless compressors, such as Zstd and LZMA, which is widely used to compress 'quantization factor' in lossy compression [6]. Here before using Zstd and LZMA in Delta-DNN, we introduce Run-Length Encoding (RLE) that efficiently records the duplicate bytes

[2] Andrew Ng. https://www.deeplearning.ai/



(a) VGG-16 (b) ResNet101 (c) GoogLeNet

(d) EfficientNet (e) MobileNet (f) ShuffleNet

Figure 3: Inference accuracy of the last model files on six neural networks using Delta-DNN with different error bounds.

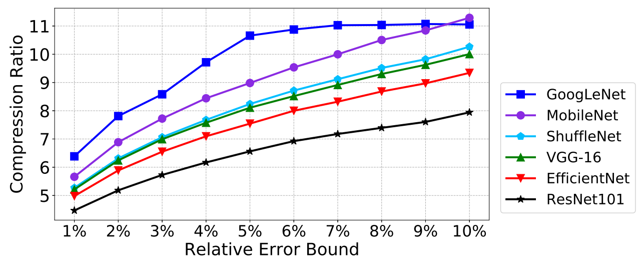


Figure 4: Compression ratio of Delta-DNN using different relative error bounds on six neural networks.

for quick space saving. This is because the delta data are calculated from the very similar floats of the neighboring neural networks, which is very compressible.

Figure 5 shows the compression ratio of four types of compressors in Delta-DNN on six DNNs, which suggests RLE+ LZMA achieves the highest compression ratio than others. Thus we adopt this hybrid approach combining RLE and LZMA to compress the lossy delta data into the compressed binary file in Delta-DNN.

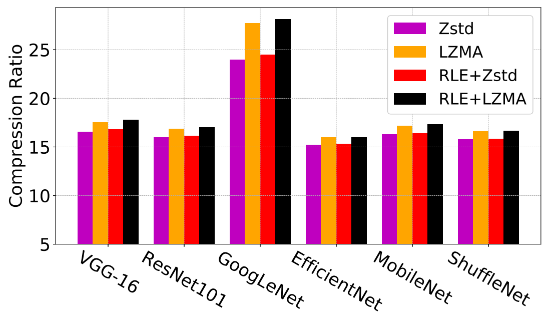


Figure 5: Compression ratios of Delta-DNN running 4 compressors (to process the lossy delta data) in error bound 10%.

To recover the target network, as also shown in Figure 2, we will decompress the compressed binary file by LZMA to get the delta data M_i, to recover all the parameters in the target network according to the reference network and the delta data M_i (as illustrated in Equation (2) in Subsection 3.2), which is very fast.

4 TYPICAL APPLICATION SCENARIOS

In this section, we introduce two typical application scenarios for delta compressing the deep neural networks (or called model) using Delta-DNN: optimizing the network transmission and saving the storage space for DNNs.

4.1 Optimizing Network Transmission for DNNs

In view of the effect of Delta-DNN framework, calculating and compressing the delta data between neighboring neural networks, we consider one of the current popular ways: DNNs are trained on the server and deployed locally on the client [21, 29] (such as mobile device [53] and IoT device [22]). Therefore, the process of transferring the newly trained model (i.e., the latest version of the deep neural network) from the server to the client is usually similar to that of the software updating on the smart devices. It can be regarded as packaging the DNN (or called the model) into an updating package (usually using lossless compressors to reduce size) on the server and then transmitting it to the target devices via network.

In this scenario, the network of the resource-constrained clients is usually a system bottleneck when updating DNNs [3, 9, 18]. Thus, applying Delta-DNN in this scenario, as shown in Figure 6, can optimize the network transmission for DNNs by compression. Specifically, we deploy Delta-DNN on the server, calculate and compress the lossy delta data of the target model along with the reference model, and then transmit the compressed file to the client devices.

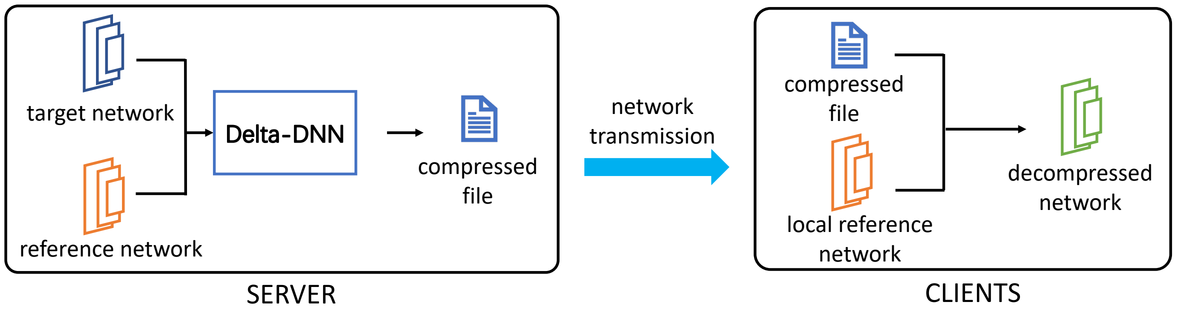


Figure 6: Delta-DNN for reducing network transmission.

On the client devices, Delta-DNN is also deployed for decompression, and the delta data will be decompressed for recovering the target model along with the reference model.

In this scenario, the client only needs to download the full network model package when installing the model at the first time, and when updating, it only needs to download the lossy delta data generated by Delta-DNN (from the server), which can efficiently reduce the network overhead of transferring the updated model. And we also evaluate Delta-DNN performance on this scenario in Section 5.3. Note that here we should always guarantee that: ① the target network always remains the same as the traditional way (i.e., without using Delta-DNN) during training; ② the reference network on the clients (used for decompression) is the same as the reference network on the server (used for compression, it is also a lossy version). Therefore, the accuracy loss is always under control.

4.2 Saving Storage Space for DNNs

In many applications, neural networks are trained with dynamically growing datasets (i.e., size of training data will be enlarged in applications with time) or different tasks (e.g., Transfer Learning [54]), and the network model needs to be continuously trained and updated. Thus multiple snapshots or versions of DNNs are saved in the training process for the possible purposes of avoiding over-fitting [1], Transfer Learning, and Incremental Learning [2].

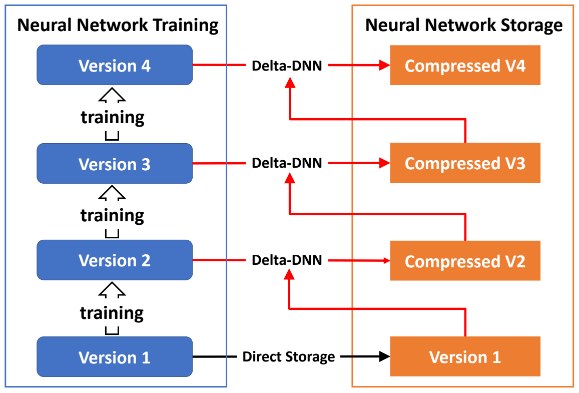


Figure 7: Delta-DNN for reducing storage cost.

In this scenario, with Delta-DNN, the original version of the neural network will be fully saved, and the following versions will be saved as a lossy delta data which is much smaller than the full version, and thus the storage space will be greatly reduced, which is shown in Figure 7. The left part in Figure 7 represents the model training/fine-tuning processes and the model storage process in the traditional way; The right part indicates that Delta-DNN is applied. After processing by Delta-DNN, only the lossy delta data of the two neighboring versions will be stored, instead of the full model, and thus the consumption of storage for DNNs can be greatly reduced, which will be evaluated in Section 5.4.

When recovering a model for retraining/fine-tuning, Delta-DNN just needs to process the lossy delta data along with the reference model. Note that there is a delta "chain" [42, 51] in this scenario as shown in Figure 7: to recover the target network *v4*, Delta-DNN needs to first recover the reference network *v3* and thus another reference network *v2*, which is time-consuming. To guarantee the recovering performance of the latest version, backward encoding (using the latest one as the reference network) or hop encoding (selectively and fully storing some networks), which are widely used in the traditional delta compression based storage systems [51, 52].

4.3 Discussion

Using Delta-DNN without pre-known accuracy: In some practical use cases, we need to apply DNNs on a new task, and the accuracy in this situation is unknown to us. In this case, although the actual accuracy is not available, the inference accuracy in training and testing can be acquired, and an 'acceptable' loss of accuracy can be defined and adjusted based on the testing inference accuracy. On the other hand, the error bound range in the paper (configured as 1%-10%) is just for the demonstration purpose. Respecting the individual use cases, a user can set and adjust their error bound range according to their applications to obtain an optimal result.

Parallelizing the compression process in Delta-DNN: In subsection 3.3, the compressing process needs to acquire the inference accuracy on each error bound, and the process is a loop, which is shown in Algorithm 1. Therefore, the time cost is nearly to $O(n)$, where n is the number of candidate error bounds. To accelerate the compression process, we can run Algorithm 1 parallel with multi-thread or multi-progress.

Compression overhead of Delta-DNN on larger datasets: In subsection 3.3, Algorithm 1 needs to compress and decompress the model several times to explore the effects of error bounds, and pick up the best one. In this process, the main overhead is to obtain the inference accuracy on each error bound after compression and decompression. Obtaining the inference accuracy means running decompressed DNNs on a user-specified test dataset, and the overhead is linear related to the size of the dataset. To reduce this overhead, there are some solutions to effectively reduce the size of test datasets, like Dataset Distillation [48] or analyze the distribution of datasets.

5 PERFORMANCE EVALUATION

In this section, we evaluate our proposed Delta-DNN framework compared with state-of-the-art compressor Zstd, LZMA, SZ (also used in DeepSZ [15]), on six popular DNNs.

5.1 Experimental Setup

We conduct our experiments on an Ubuntu server with an Intel Xeon Gold 6130 processor (with 128 GB of memory) and a NVIDIA TITAN RTX GPU (with 24 GB of memory).

We implement Delta-DNN based on the known Pytorch deep learning framework [34]. Six popular DNNs[3] are used in our evaluation: VGG-16 [40], ResNet101 [10], GoogLeNet [43], EfficientNet [46], MobileNet [38], and ShuffleNet [55]. We train each neural network on CIFAR-10 dataset [17] by SGD with Batch Normalization and momentum (learning rate=0.01, momentum=0.9, and weight decay=1e-4). These neural networks and the dataset are commonly used in amounts of studies for DNNs compression [29, 50].

In Delta-DNN framework, we set the default relative error bounds as 0.1% and the optimized error bound is selected from 1%~10% according to Algorithm 1 (where α and β are all set to 0.5). In terms of neural network preserving, we generally follow the best inference accuracy method, that is, during training, the epoch will be saved if the inference accuracy of the neural network has been improved on the test dataset, or within a fixed epochs interval (e.g., save the first one for every 20 epochs).

Three state-of-the-art compressors are evaluated for comparison in this section: Zstd, LZMA, and SZ, which are used directly on the floats of the evaluated DNNs. Zstd is short for Zstandard [57], which is developed recently by Facebook, and it consists of dictionary coding and entropy coding techniques. LZMA is a compression approach focusing on the compression ratio, also known as 7zip [16]. SZ is an open-source project for compressing large amounts of scientific data, and is also used in DeepSZ [15] for compressing floats after using the pruning techniques. Among them, Zstd and LZMA belong to the lossless compression, while SZ is for lossy compression (similar to Delta-DNN). To get the optimal error bound for SZ, we also use Algorithm 1 for SZ to select the best error bound from 1%~10%, with the same configuration as Delta-DNN.

5.2 Compression Performance of Delta-DNN

In this subsection, we evaluate the overall compression performance of Delta-DNN, mainly using two metrics: compression ratio and inference accuracy of DNNs. In the evaluation, the acceptable relative inference accuracy loss for Delta-DNN and SZ is set to 0.2%, which reflects the maximal tolerable accuracy loss using lossy compression, as discussed in Subsection 3.3.

Table 1 shows the compression ratio results of the four compressors on six popular DNNs: compressing the last epoch of the neural network that has the highest inference accuracy. Among them, Zstd and LZMA are lossless compression algorithms, and both of them do not cause any changes in inference accuracy. As the lossy compressors, the network inference accuracy results of Delta-DNN and SZ, are also shown in this table. For comparison to the original DNN accuracy before lossy compression, the percentage differences (in parentheses) are also calculated in the table, where '-' means accuracy loss and '+' means accuracy gain.

As shown in Table 1, Delta-DNN achieves the highest compression ratio while keeping the inference accuracy less than 0.2%. SZ is the second-best compressor in Table 1, using a smaller error bound and obtains a slightly higher accuracy loss than Delta-DNN. This is because: the to-be-compressed data in our Delta-DNN framework, namely, the float pairs in the neighboring networks, are more compressible than that in the SZ framework, i.e., the neighboring floats in the data arrays. Meanwhile, LZMA and Zstd achieve the lowest compression ratio, all less than 1.1, due to the known reason that the floating-point numbers with the random ending mantissa bits are difficult to be compressed (by lossless compressors).

Table 2 further studies the compression ratio and inference accuracy of Delta-DNN and the other three compressors on different epochs of VGG-16. Here the epochs of VGG-16 (i.e., the stable versions) are saved only if the inference accuracy has been improved over the last saved epoch on the datasets during training. The results shown in this table are generally consistent with that in Table 1, which suggests Delta-DNN achieves the highest compression ratio and the comparable inference accuracy in all the tested five epochs. Note that the results of other DNNs are similar to those of VGG-16 and are omitted due to space limit.

In all, by exploiting the floats similarity existing in the neighboring networks for the efficiently lossy delta compression, Delta-DNN achieves about $2\times \sim 10\times$ higher compression ratio compared with the state-of-the-art approaches, LZMA, zstd, and SZ, while keeping the inference accuracy loss smaller than 0.2%.

5.3 Case 1: Optimizing Network Transmission

Network transmission of DNNs is a widely used application scenario for deep learning. In this subsection, we use the statistical global average network bandwidth from the *SPEEDTEST* tool [4] as the test bandwidth, to evaluate the transmission time required for DNNs, after compression by Delta-DNN and other approaches.

As discussed in Subsection 4.1, Delta-DNN is designed to reduce network transmissions by delta compressing the neighboring neural networks, and then sending the compressed networks from server to clients. Here the server mainly uses the wired network to transfer models to clients, and the clients can use both the wired or wireless network to download models from the server. So the network configuration for this case includes uploading and downloading models over the wired network, and downloading models over the wireless network. And the *SPEEDTEST* tool provides the global average network bandwidth in January 2020: the upload bandwidth of wired broadband is $40.83 Mbps$, the download bandwidth of wired broadband is $74.32 Mbps$, and the download bandwidth of wireless broadband is $31.95 Mbps$.

Figure 8 shows the time cost of the network transmission of the six DNNs after compression by the four compressors, i.e., running

[3]https://github.com/kuangliu/pytorch-cifar.

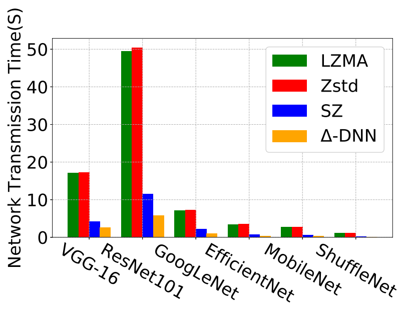
[4]SPEEDTEST. https://www.speedtest.net/global-index

Table 1: Compression ratio and inference accuracy of LZMA, Zstd, SZ, and Delta-DNN (Δ-DNN).

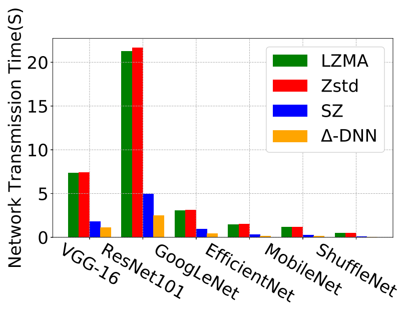
Networks	Original Size	**Compression Ratio** (and the error bound)				**Inference Accuracy** (and the differences)		
		LZMA	**Zstd**	**SZ**	**Δ-DNN**	**Original**	**SZ**	**Δ-DNN**
VGG-16	56.2 MB	1.096	1.088	4.415 (7%)	7.394 (8%)	92.45%	92.31% (–0.15%)	92.32% (–0.15%)
ResNet101	162.6 MB	1.098	1.078	4.192 (5%)	9.341 (10%)	93.05%	92.87% (–0.19%)	93.44% (+0.42%)
GoogLeNet	23.6 MB	1.097	1.078	3.565 (2%)	7.811 (2%)	94.95%	94.88% (–0.07%)	94.95% (+0.00%)
EfficientNet	11.3 MB	1.099	1.078	3.204 (1%)	10.266 (10%)	84.82%	84.76% (–0.07%)	84.88% (+0.07%)
MobileNet	8.9 MB	1.101	1.077	3.788 (3%)	9.627 (9%)	92.68%	92.57% (–0.12%)	93.16% (+0.52%)
ShuffleNet	3.5 MB	1.097	1.076	3.192 (1%)	11.291 (10%)	86.29%	86.19% (–0.12%)	86.18% (–0.13%)

Table 2: Inference accuracy and compression ratio for the last 5 epochs of VGG-16 with and without Delta-DNN.

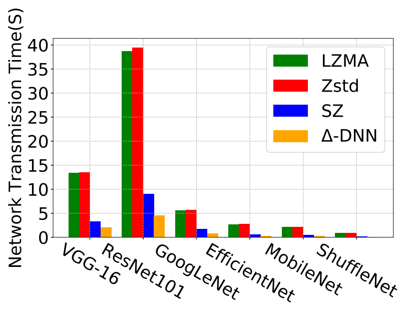
Epochs	**Compression Ratio** (and the error bound)				**Inference Accuracy** (and the differences)		
	LZMA	**Zstd**	**SZ**	**Delta-DNN**	**Original**	**SZ**	**Delta-DNN**
740	1.090	1.081	4.154 (5%)	5.702 (6%)	91.19%	91.04% (–0.16%)	91.25% (+0.07%)
744	1.090	1.081	3.959 (4%)	7.130 (7%)	91.99%	91.81% (–0.20%)	91.92% (–0.08%)
747	1.089	1.080	4.277 (6%)	6.626 (5%)	92.13%	92.09% (–0.04%)	92.07% (–0.07%)
761	1.089	1.080	3.775 (3%)	6.717 (10%)	92.23%	92.30% (+0.08%)	92.16% (–0.08%)
765	1.088	1.079	4.310 (6%)	7.394 (8%)	92.45%	92.44% (–0.01%)	92.32% (–0.14%)



(a) Mobile Broadband Downloading



(b) Fixed Broadband Downloading



(c) Fixed Broadband Uploading

Figure 8: Network time cost on six DNNs after compression using Delta-DNN, SZ, LZMA, and Zstd.

the compression approaches on the server/client architecture shown in Figure 6. It can be seen that Delta-DNN significantly reduces the network consumption of six neural networks, regardless of the network bandwidth configurations. This is because the Delta-DNN-compressed networks are about 7× ∼ 11× smaller than the size before compression. Meanwhile, Zstd, LZMA, and SZ take more time for network transmission, as shown in Figure 8, which is due to their lower compression ratio on DNNs.

5.4 Case 2: Saving Storage Space

In this subsection, we evaluate the performance of the Delta-DNN framework in reducing the storage overhead in neural network training where many epochs are stored. We compare Delta-DNN with other compressors on the two metrics: the DNNs' inference accuracy and storage space overhead.

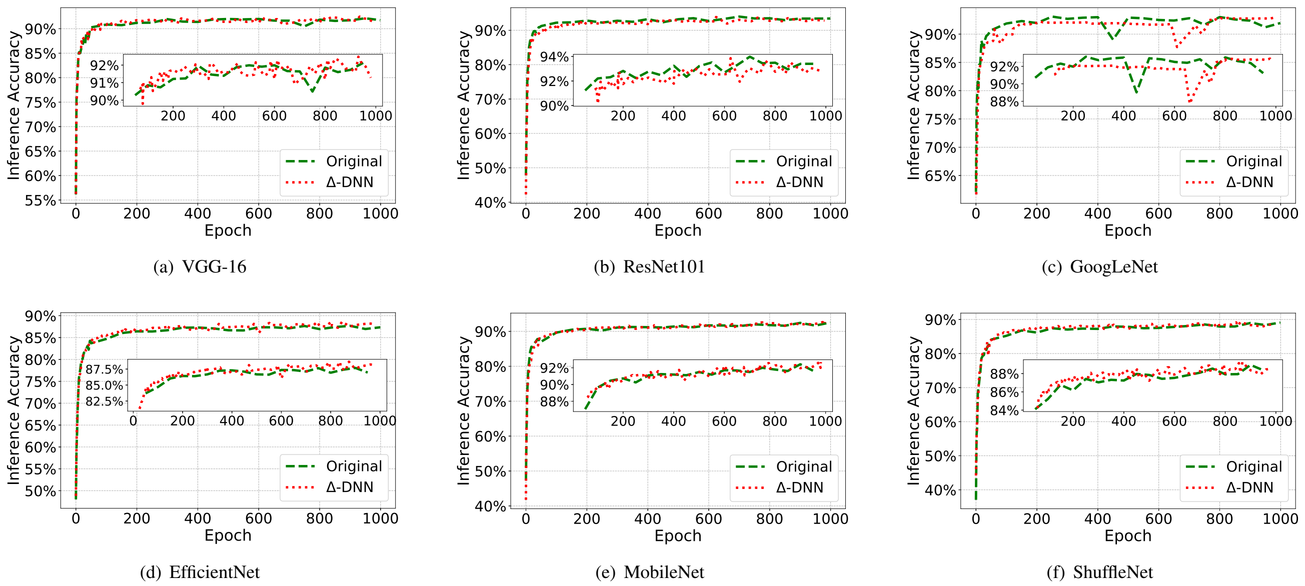
Table 3 shows the total occupied storage space, the total compression ratio, and the average accuracy loss on six neural networks in training, while using Delta-DNN. Since many versions of networks in training are saved in this case, the storage sizes are much larger than one network. However, Delta-DNN can effectively reduce the storage size of the six DNNs by 5× ∼ 10× while the average inference accuracy loss is all less than 0.001%. Note that the compression

Table 3: Storage space consumption in training 6 DNNs before and after using Delta-DNN. In the last column of the accuracy loss results, '-' denotes the inference accuracy gain.

Network	Epochs	Total Size		Comp. Ratio	Accuracy Loss
		Original	Δ-DNN		
VGG-16	95	5.21 GB	693 MB	7.702	-0.0003%
ResNet101	89	14.1 GB	2.18 GB	6.488	-0.0015%
GoogLeNet	83	1.91 GB	191 MB	10.259	-0.0009%
EfficientNet	110	1.21 GB	208 MB	5.946	0.0001%
MobileNet	115	1.00 GB	140 MB	7.311	-0.0004%
ShuffleNet	113	391 MB	73 MB	5.302	0

ratio in this table is a little lower than that in Table 1, this is because some of the saved epochs are training with a long distance (over many epochs) to get an improved inference accuracy and thus have the less similarity to be exploited for delta compression in Delta-DNN. However, the storage consumption is still significantly reduced by Delta-DNN framework.

Figure 9 shows the comparison of inference accuracy before and after using Delta-DNN when training the six DNNs. Even they are two different DNN-training processes (using Delta-DNN or not), it still could be seen that after using Delta-DNN, the inference accuracy



(a) VGG-16 (b) ResNet101 (c) GoogLeNet

(d) EfficientNet (e) MobileNet (f) ShuffleNet

Figure 9: Inference accuracy on different epochs of DNNs before and after using Delta-DNN (Δ-DNN).

in training is almost the same as the original before compression. Even the accuracy loss from Delta-DNN exists in the earlier epochs, but this loss has almost no impact on the following epochs, whose inference accuracy loss is always minimal, as shown in this figure (the average accuracy loss is also shown in Table 3). Note that the inference accuracy may decrease in the training neural networks in Figure 9 (c), this is because they belong to two training processes of mini-batch gradient descent. The same phenomenon can be also observed when repeating the same training process twice. And Delta-DNN guarantees the accuracy loss within a user-specified limit but is not responsible for the convergence of DNNs.

6 CONCLUSION AND FUTURE WORK

In this paper, we propose a novel delta compression framework for deep neural networks, called Delta-DNN, which can significantly reduce the size of DNNs by exploiting the floats similarity existing in neighboring networks in training. Specifically, we observe that large amounts of parameters are slightly changed in DNN training. And inspired by the delta compression technique used in the backup storage area, we calculate the lossy delta data between the neighboring networks and then use the lossless compressors to further reduce the delta data. The high compression ratio of Delta-DNN is due to the very compressible lossy delta data in our framework that exploits floats similarity. Our evaluation results on six popular DNNs suggest Delta-DNN achieves $2\times \sim 10\times$ higher compression ratio compared with Zstd, LZMA, and SZ approaches, while keeping the DNNs' inference accuracy loss smaller than 0.2%.

In our future work, we plan to further improve the compression ratio of Delta-DNN combining other model compression techniques (such as pruning and quantization) and evaluating it on more DNNs. Furthermore, we will continue to extend Delta-DNN framework into more scenarios, such as deep learning in the distributed systems, to further reduce the storage and network overheads.

ACKNOWLEDGEMENTS

We are grateful to the anonymous reviewers for their insightful comments and constructive suggestions on this work. This research was partly supported by NSFC No. 61972441, No. 61672186, and No. 61872110, the Shenzhen Science and Technology Program under Grant No. JCYJ20190806143405318 and JCYJ20190806142601687, Key Technology Program of Shenzhen, China under Grant No. JSGG20170823152809704, Key-Area R&D Program for Guangdong Province under Grant No. 2019B01013600, Guangdong Basic and Applied Basic Research Foundation No. 2019A1515110475.

The work of Sian Jin and Dingwen Tao was partly supported by the U.S. National Science Foundation under Grants OAC-1948447 and OAC-2003624, and the U.S. Exascale Computing Project (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.

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the funding agencies. Wen Xia is the corresponding author (xiawen@hit.edu.cn).

REFERENCES

[1] Rich Caruana, Steve Lawrence, and C Lee Giles. 2001. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. In *Advances in neural information processing systems*. 402–408.

[2] Francisco M Castro, Manuel J Marín-Jiménez, Nicolás Guil, Cordelia Schmid, and Karteek Alahari. 2018. End-to-end incremental learning. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 233–248.

[3] Yuntao Chen, Naiyan Wang, and Zhaoxiang Zhang. 2018. Darkrank: Accelerating deep metric learning via cross sample similarities transfer. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[4] Gobinda G Chowdhury. 2003. Natural language processing. *Annual review of information science and technology* 37, 1 (2003), 51–89.

[5] Peter Deutsch et al. 1996. GZIP file format specification version 4.3.

[6] Sheng Di and Franck Cappello. 2016. Fast error-bounded lossy HPC data compression with SZ. In *2016 ieee international parallel and distributed processing symposium (ipdps)*. IEEE, 730–739.

[7] Peiyan Dong, Siyue Wang, Wei Niu, Chengming Zhang, Sheng Lin, Zhengang Li, Yifan Gong, Bin Ren, Xue Lin, Yanzhi Wang, and Dingwen Tao. 2020. RTMobile: Beyond Real-Time Mobile Acceleration of RNNs for Speech Recognition. *arXiv preprint arXiv:2002.11474* (2020).

[8] Mike Dutch. 2008. Understanding data deduplication ratios. In *SNIA Data Management Forum*. 7.

[9] 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*.

[10] 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.

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

[12] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized neural networks. In *Advances in neural information processing systems*. 4107–4115.

[13] David A Huffman. 1952. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE* 40, 9 (1952), 1098–1101.

[14] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. 2014. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*.

[15] 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 *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing*. 159–170.

[16] Dominik Kempa and Simon J Puglisi. 2013. Lempel-Ziv factorization: Simple, fast, practical. In *2013 Proceedings of the Fifteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*. SIAM, 103–112.

[17] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. 2014. The cifar-10 dataset. *online: http://www. cs. toronto. edu/kriz/cifar. html* 55 (2014).

[18] Nicholas D Lane, Sourav Bhattacharya, Petko Georgiev, Claudio Forlivesi, Lei Jiao, Lorena Qendro, and Fahim Kawsar. 2016. Deepx: A software accelerator for low-power deep learning inference on mobile devices. In *2016 15th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. IEEE, 1–12.

[19] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436–444.

[20] Dawei Li, Xiaolong Wang, and Deguang Kong. 2018. Deeprebirth: Accelerating deep neural network execution on mobile devices. In *Thirty-second AAAI conference on artificial intelligence*.

[21] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. 2016. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*.

[22] He Li, Kaoru Ota, and Mianxiong Dong. 2018. Learning IoT in edge: Deep learning for the Internet of Things with edge computing. *IEEE network* 32, 1 (2018), 96–101.

[23] Sihuan Li, Sheng Di, Xin Liang, Zizhong Chen, and Franck Cappello. 2018. Optimizing lossy compression with adjacent snapshots for N-body simulation data. In *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 428–437.

[24] Zhaoqi Li, Yu Ma, Catalina Vajiac, and Yunkai Zhang. 2018. Exploration of Numerical Precision in Deep Neural Networks. *arXiv preprint arXiv:1805.01078* (2018).

[25] Xin Liang, Sheng Di, Sihuan Li, Dingwen Tao, Zizhong Chen, and Franck Cappello. [n.d.]. Exploring Best Lossy Compression Strategy By Combining SZ with Spatiotemporal Decimation.

[26] 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. In *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 438–447.

[27] Peter Lindstrom. 2014. Fixed-rate compressed floating-point arrays. *IEEE Transactions on Visualization and Computer Graphics* 20, 12 (2014), 2674–2683.

[28] Peter Lindstrom. 2017. *Error distributions of lossy floating-point compressors*. Technical Report. Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States).

[29] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. 2017. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE International Conference on Computer Vision*. 2736–2744.

[30] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. 2018. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*.

[31] Mengting Lu, Fang Wang, Dan Feng, and Yuchong Hu. 2019. A Read-leveling Data Distribution Scheme for Promoting Read Performance in SSDs with Deduplication. In *Proceedings of the 48th International Conference on Parallel Processing*. 1–10.

[32] Yao Lu, Guangming Lu, Jinxing Li, Yuanrong Xu, Zheng Zhang, and David Zhang. 2020. Multiscale conditional regularization for convolutional neural networks. *IEEE Transactions on Cybernetics* (2020).

[33] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. 2016. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*.

[34] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*. 8024–8035.

[35] Igor Pavlov. 1998. The Algorithm: Lempel-Ziv-Markov Chain.

[36] Majid Rabbani. 2002. JPEG2000: Image compression fundamentals, standards and practice. *Journal of Electronic Imaging* 11, 2 (2002), 286.

[37] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision*. Springer, 525–542.

[38] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4510–4520.

[39] Jerome M Shapiro. 1993. Embedded image coding using zerotrees of wavelet coefficients. *IEEE Transactions on signal processing* 41, 12 (1993), 3445–3462.

[40] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

[41] 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.

[42] Torsten Suel, Nasir Memon, and Khalid Sayood. 2002. Algorithms for delta compression and remote file synchronization. *Lossless Compression Handbook*.

[43] 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.

[44] Richard Szeliski. 2010. *Computer vision: algorithms and applications*. Springer Science & Business Media.

[45] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. 2014. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1701–1708.

[46] Mingxing Tan and Quoc V Le. 2019. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*.

[47] 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 (IPDPS)*. IEEE, 1129–1139.

[48] Tongzhou Wang, Jun-Yan Zhu, Antonio Torralba, and Alexei A Efros. 2018. Dataset distillation. *arXiv preprint arXiv:1811.10959* (2018).

[49] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. 2004. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing* 13, 4 (2004), 600–612.

[50] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. 2016. Learning structured sparsity in deep neural networks. In *Advances in neural information processing systems*. 2074–2082.

[51] Wen Xia, Hong Jiang, Dan Feng, Fred Douglis, Philip Shilane, Yu Hua, Min Fu, Yucheng Zhang, and Yukun Zhou. 2016. A comprehensive study of the past, present, and future of data deduplication. *Proc. IEEE* 104, 9 (2016), 1681–1710.

[52] Lianghong Xu, Andrew Pavlo, Sudipta Sengupta, and Gregory R Ganger. 2017. Online deduplication for databases. In *Proceedings of the 2017 ACM International Conference on Management of Data*. 1355–1368.

[53] Lei Yang, Jiannong Cao, Zhenyu Wang, and Weigang Wu. 2017. Network aware multi-user computation partitioning in mobile edge clouds. In *2017 46th International Conference on Parallel Processing (ICPP)*. IEEE, 302–311.

[54] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. 2014. How transferable are features in deep neural networks?. In *Advances in neural information processing systems*. 3320–3328.

[55] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. 2018. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 6848–6856.

[56] Jacob Ziv and Abraham Lempel. 1977. A universal algorithm for sequential data compression. *IEEE Transactions on information theory* 23, 3 (1977), 337–343.

[57] Zstandard. 2018. Fast real-time compression algorithm. http://facebook.github.io/zstd/