Structured Weight Unification and Encoding for Neural Network Compression and Acceleration

Wei Jiang
Tencent America LLC.
Palo Alto, CA
vwjiang@tencent.com

Wei Wang
Tencent America LLC.
Palo Alto, CA
rickweiwang@tencent.com

Shan Liu
Tencent America LLC.
Palo Alto, CA
shanl@tencent.com

Abstract

We investigate structured joint weight unification and weight encoding to compress deep neural network models for reduced storage and computation. A structured weight unification method is proposed, where weight coefficients are unified according to a hardware-friendly structure, so that the unified weights can be effectively encoded and the inference computation can be accelerated. Our method can be seen as a generalization of structured weight pruning, where we unify weights of a selected structure to share some value instead of removing them. A 3D pyramid-based encoding method is further proposed to team up with the structurally learned weights, providing a systematic solution for compressing neural network models while preserving the network capacity and the original prediction performance. Also, we develop a training framework to iteratively optimize the subproblems of weight unification and target prediction, which ensures the unification rate with little prediction loss. Experiments over several benchmark models and datasets of different tasks demonstrate the effectiveness of our approach.

1. Introduction

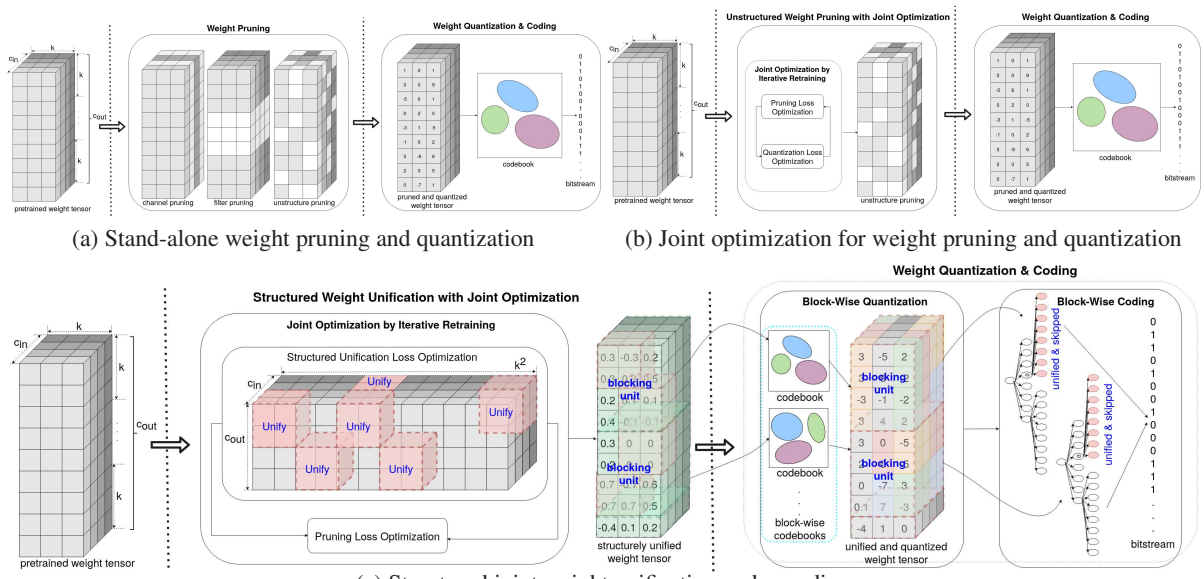
Deep Neural Networks (DNNs) have achieved great success in solving a wide range of tasks for computer vision, natural language processing, *etc.* The large model capacity of the deep network structures with huge amount of parameters leads to high prediction performance, but also makes DNN models too expensive to use in practice, especially for mobile and on-device applications with strong limitations on storage, computation power, and energy consumption. It has drawn great attention how to reduce the costs of using DNNs in academia and industry. A special group is also formed by the internatioanl standardization organization MPEG addressing this issue [18].

Active research has been conducted in the past years to compress large DNN models. The overall target is to reduce the size of the model (*i.e.*, the required storage) and to accelerate inference, without sacrificing much the performance of the original task (*e.g.*, classification accuracy). Effective solutions usually require multidisciplinary knowledge from machine learning, computer architecture, hardware design, *etc.*, and great progress has been made using different techniques, including weight pruning [8, 9, 11, 15, 16, 24], weight quantization [25, 26, 27], low-rank factorization [13], and knowledge distillation [12].

Among all the efforts, weight pruning and weight quantization are the most popular directions. Weight pruning aims at removing unimportant weight coefficients and reducing the redundancy in network connections. The pioneer work of [8] proposed to prune pretrained weight tensors followed by a stand-alone encoding process comprising quantization and entropy coding, as shown in Fig. 1a. Several extensions have also been developed [5, 7]. Although high compression rate can be achieved with little prediction loss, such unstructured weight pruning methods can not reduce inference computation (sometimes even worsen the problem) in general [21, 23], due to the random memory access caused by the unstructured sparsity in the pruned weight matrices.

To overcome such drawbacks, the structured weight pruning methods deliberately induce sparsity according to some hardware-friendly patterns [11, 16, 23, 24, 28]. Unimportant structures such as channels, filters, or layers are removed by minimizing the pruning loss measuring the changes of weights, activations, *etc.* However, removing entire weight structures usually causes large prediction performance drop, especially for models like MobileNet [22] that are designed to be highly efficient already.

Weight quantization [25, 26, 27] aims at reducing the number of bits to represent the weight coefficients of DNNs. Both storage and inference computation can be reduced propotionally to weight precision naturally. However, training with quantized weights usually requires good gradient estimation and stability control, due to the instability and difficulty in back-propagation caused by integer weights.



(a) Stand-alone weight pruning and quantization

(b) Joint optimization for weight pruning and quantization

(c) Structured joint weight unification and encoding

Figure 1: Framework of our approach and prior weight pruning and weight quantization methods. Our weights are structurally unified to help the following encoding process as well as speeding up the inference computation. A 3D-octtree-based encoding method is also developed to team up with the structurally learned weights, providing a compression system for both storage and computation reduction.

Jointly exploring weight pruning and quantization can potentially improve the performance of individual solutions. Naiive weight learning by optimizing the joint loss, however, is quite difficult, due to similar reasons in weight quantization. The recent work of [21] proposed an unstructured solution (as shown in Fig. 1b) to the problem by decomposing it into unstructured weight pruning and quantization subproblems whose losses can be iteratively optimized through Alternating Direction Methods of Multipliers (ADMM) optimization. Actual weight quantization is conducted after the training process to avoid learning with quantized weights. State-of-the-art compression rate has been achieved with little accuracy degradation.

In this paper, we go one step further. We study structured joint weight unification and weight encoding (*i.e.*, quantization as well as the following entropy coding). Similar to [21], we iteratively optimize the subproblems of weight unification and weight retraining in separate steps, but in a structured fashion (as shown in Fig. 1c). Our structured weight unification approach unifies weights within a certain structure in a way to facilitate the following encoding. Also, the unified weight structure is designed to accelerate the underlying convolution operation so as to speed up the inference computation. A hardware-friendly 3D pyramid-based encoding method is further developed to team up with the structurally learned weights, resulting in a systematic solution for both storage and computation reduction.

From another perspective, our weight unification can be seen as a generalization of the weight pruning methods, where we set selected weights to a unified value instead of 0. The main advantage is that by unifying weight structures instead of removing them, the network capacity (so as the prediction performance) can be better preserved. An iterative two-step training framework is further developed to effectively ensure the unification rate with little prediction performance loss. The framework is flexible to accomodate various encoding methods and unification structures.

We evaluate our algorithm over several benchmark models and datasets of different tasks, including ImageNet classification, end-to-end image compression, and audio classificatoin. Our experiments are consistent with the neural network compression standardization efforts of MPEG [18, 19], which aims at defining a compressed representation for trained DNNs with scalability and generality. Experimental results are quite promising. For example, for ImageNet classificatoin, with only 2% top-1 and top-5 accuracy loss, we got 14× and 9× compression rate as well as 0.83 and 0.06 gigaMACs inference speed up using ResNet-50 and MobileNet-V2, respectively. For image compression, we compress the Autoencoder by 6× with 6 million-MACs inference speed up. We also compressed audio classification network by 20× with 23 millionMACs inference speed up. Moreover, when the compression rate is small, we often can improve the prediction performance of the original pretrained model. Such phenomena, combined with the good compression rate achieved over already compact models like MobileNet-V2, further confirm that our structured joint approach can effectively reduce the redundancy in DNNs while maintaining the network learning capacity.

The rest of the paper is organized as follows. Section 2 reviews related prior arts. Section 3 describes our weight unification method. Section 4 introduces our 3D pyramid-based encoding. Section 5 shows experimental results, and Section 6 concludes our paper with some discussions.

2. Related Works

Active research has been conducted recently to compress large DNN models for efficient computation. The target is to simultaneously reduce the model size and accelerate inference computation, with small accuracy loss. Among all the efforts, weight pruning and weight quantization are two major promising directions. In this section we briefly go over some prior arts related to our work. More comprehensive summaries can be found in surveys like [3].

2.1. Weight pruning

Weight pruning methods reduce the redundancy of network parameters by reducing the number of weight coefficients. The pionior work of [9] achieved about $10\times$ reduction on classic networks like AlexNet and VGG by iteratively removing weak weight connections in a heuristic way. Several extensions like [5, 7] have been developed ever since. The basic idea is to place regularization over weight coefficients, *e.g.*, L_0 or L_1 norm, to promote sparsity of the learned weights. However, the sprase weight coefficients obtained by such unstructured pruning methods are usually irregular, i.e., the pruned out weights are scattered irregularly in weight tensors, resulting in very limited benefit in computation acceleration.

The convolutional computation in DNN is commonly implemented as GEneral Matrix Multiplication (GEMM) by reshaping the weight tensors and input/output feature tensors to matrices [4]. In this context, the structured weight pruning approaches can better pursue both storage and computation efficiency than the unstructured methods. The main idea is to remove weight coefficients in a structured way that can benefit the GEMM computation. For example, the $L_{2,1}$ norm was used on convolutional layers to promote group-sparsity [28]. The trivial channels, filters, or layers were removed in [24] to directly reduce rows and columns of weight matrices in GEMM computation. Channel pruning [11] iteratively pruned each network layer by alternating channel selection and reconstruction. Filter pruning [16] used L_1 norm to select and prune unimportant filters.

2.2. Weight quantization

The goal of weight quantization is to reduce the number of bits to represent the model with tolerable performance degradation. This technique is usually hardware-friendly, which improves both computation and storage. From another point of view, pruned model after weight pruning can be further compressed through weight quantization. Therefore, effective weight quantization algorithms that jointly consider DNN learning, optimization computer architecture and hardware design are highly demanded by industry. Several methods have been proposed to quantize DNN weights. For example, early work like [25] proposed to quantize weights in a layer-wise scheme to reduce performance drop. An iterative quantization and retraining framework was proposed in [27] which alternated the weight quantization step and weight retraining step to achieve balanced quantization ratio and accuracy loss incrementally.

In general, DNN training with quantized weights can be challenging. Quantized weights often make the training process unstable [26], and also make back-propagation difficult. As a result, sucessful network training depends on good gradient estimation and stability control techniques.

2.3. Joint weight pruning and quantization

To fully benefit from both weight pruning and weight quantization, recent efforts start to explore joint weight pruning and quantization. The deep compression method in [8] removed the redundant connections, quantized weights, and coded the quantized weights. A regularization based on soft weight-sharing was used to obtain pruned and quantized weights in [14]. A systematic ADMM-NN algorithm was proposed in [21], which decomposed the joint optimization problem into unstructured weight pruning and weight quantization subproblems, which are solved separately by Alternating Direction Methods of Multipliers (ADMM). State-of-the-art compression rates were obtained with little accuracy loss, *e.g.*, $17\times$ weight reduction on ResNet-50. However, the method can not enforce structured weight pruning, which, when systematically combined with the optimization computer architecture and hardware design, can largely speed up inference computation.

2.4. Our motivation

In this paper, we study structured joint weight unification and encoding including both quantization and the following entropy coding. Our goal is to compress the DNN model into a compact bitstream that can reduce both storage and comptuation costs. We generalize the idea of structured weight pruning and propose a structured weight unification approach, where locally unified real values are assigned to weight coefficients in a structured way. Weight pruning can be viewed as a special case of our method where the unified real value is zero. With such a generalization, we jointly consider weight unification and weight encoding by enforcing a unified weight structure that is aligned with the GEMM computation. Furthermore, we develop an iterative retraining framework to effectively ensure the unification rate with little prediction degradation.

3. Structured Weight Unification & Encoding

We first formulate our problem. Let $\{W_i\}$ denote a set of weight coefficients of a pre-trained DNN model, where W_i represents the weights of the i-th layer. The previous weight pruning methods can be described as finding a binary mask M_i for each W_i where weights in W_i corresponding to zero

entries in M_i are set to value 0. The optimal mask M_i and weights W_i are jointly optimized in a layer-wise fashion by optimizing a joint loss function:

$$\mathcal{L}_\Theta = \mathcal{L}(\mathcal{D}|\Theta) + \alpha\mathcal{L}_p(\Theta), \quad (1)$$

where $\Theta = (\{W_i\}, \{M_i\})$ denotes the whole set of parameters to learn; $\mathcal{L}(\mathcal{D}|\Theta)$ is the original data loss over a training data set $\mathcal{D}$; $\mathcal{L}_p(\Theta)$ is the sparsity-promoting regularization over parameters Θ; and α is the hyperparameter balancing different terms.

On the other hand, the previous weight quantization methods can be described as finding a set of quantized weights $\Psi = \{\hat{W}_i\}$ so that the following loss is minimized:

$$\mathcal{L}_\Psi = \mathcal{L}(\mathcal{D}|\Psi) + \beta\mathcal{L}_q(\Psi), \quad (2)$$

where $\mathcal{L}_q(\Psi)$ is a regularization (usually non-convex) placed over quantized weight coefficients. For example, $\mathcal{L}_q(\Psi)$ can promote some desired characteristics of the quantized weights, *e.g.*, allocating more/fewer bits to more/less important weights.

Naiive joint weight pruning and quantization by optimizing both Eqn. (1) and (2) together is, unfortunately, hard and inefficient, due to the difficulty of back-propagation and unstability in training with quantized weights.

In this paper, we go one step further to investigate joint weight unification and encoding. We decompose the overall problem and solve it in separate steps while keeping in mind the potential impacts of the subproblems on each other. Inspired by [21], we conduct weight encoding outside of the weight retraining process. This avoids the instability in training and the difficulty of back-propagation caused by learning using encoded weights. In summary, for weight unification, we want to learn compact weights in a structured way that can accelerate inference computation, and at the same time, can be effectively encoded later for size reduction. For weight encoding, we select a hardware-friendly locality-sensitive encoding method that can benefit from the learned weights of the unification process.

3.1. Structured weight unification

The optimization problem of Eqn. (1) can be seen as a special case of the following general task. Given a pre-trained DNN model $\{W_i\}$, we would like to find a mask M_i and a unifier U_i for each W_i, where weights corresponding to the zero entries in the mask are set to some unified values by the corresponding unifier. Each unifier U_i carries two types of information: structural information indicating the set of structures in W_i to be unified, and unifying operations determining how to set values of items in the unification structures for W_i. For the case of weight pruning, the masked out weights by M_i are all set to value 0.

Such a generalization provides three main benefits. First, instead of setting all selected weights to 0, the non-zero weights can still contribute to the network function, and the original prediction performance can be better preserved. Second, in terms of model compression, our unifier U_i is locality-sensitive, which assigns locality-sensitive unified values to weights in a structured way to ensure that the learned model can be effectively compressed by further quantization and entropy coding process. Third, the locality-sensitive unifier can also set weights to have a structure that aligns with the underlying GEMM operation, so that the inference can be effectively accelerated.

We jointly consider all aspects and take a general mixed approach: some weights are pruned (set to 0), and some weights are structurally unified, to find an optimal compact model, which can preserve the original prediction target, can be effectively encoded later, and is fast for inference.

Specifically, we find the optimal set of parameters $\Phi = (\{W_i\}, \{M_i\}, \{U_i\})$ by optimizing the following loss in an alternative iterative process:

1. Find the unifier $\{U_i\}$ by minimizing the unification loss $L_u(\Phi)$, and then unify $\{W_i\}$ using $\{U_i\}$
2. Update un-unified weights in $\{W_i\}$ and masks $\{M_i\}$ by minimizing the joint loss:

$$\mathcal{L}_\Phi = \mathcal{L}(D|\Phi) + \alpha\mathcal{L}_p(\Phi). \quad (3)$$

$\mathcal{L}_p(\Phi)$ is the sparsity-promoting regularization over Φ similar to that in Eqn. (1). $\mathcal{L}_u(\Phi)$ is the structural unification loss over Φ to promote effective encoding and inference acceleration of the learned model.

3.1.1 GEMM with unification structures

Let's consider the general multiplication process of a left-hand-side $m \times r$ matrix $\mathbf{A}$ and a right-hand-side $r \times n$ matrix $\mathbf{B}$. Structured weight pruning removes rows and/or columns of the matrices to skip multiplications in the GEMM computation. To accelerate computation of our unified weights, we propose a GEMM operation based on our unification strcuture so that we can skip multiplications with or without removing entire rows or columns. When the weights of a unification structure are set to zero, we completely skip the multiplication of that structure. When the weights of a unification structure are unified as non-zero values, we reduce the number of multiplication operations by sharing the multiplication outputs within that structure. Therefore, the previous row/column removal can be seen as a special case of our GEMM operation, where the unification structure is row/column and weights are all set to 0.

We observe that if p coefficients in a row of $\mathbf{A}$ share the same absolute value, they share one multiplication operation and $p-1$ multiplication operations can be skipped. For example, if the first p coefficients in a row of $\mathbf{A}$ have the same absolute value, the output of these coefficients can be generated with only one multiplication operation:

$$\mathbf{O}_{ij} = |\mathbf{A}_{il_1}| \cdot \sum_{l_1=0}^{p-1} J_{l_1 j} + \sum_{l_2=p}^{r-1} \mathbf{A}_{il_2} \cdot \mathbf{B}_{l_2 j},$$

$$J_{l_1 j} = \begin{cases} \mathbf{B}_{l_1 j}, & \text{if } \mathbf{A}_{il_1} > 0 \\ 0, & \text{if } \mathbf{A}_{il_1} = 0 \\ -\mathbf{B}_{l_1 j}, & \text{if } \mathbf{A}_{il_1} < 0 \end{cases}$$

where $|\cdot|$ takes the absolute of a number. If two rows of matrix **A** are identical, the entire computation of the second row can be skipped by reusing the output computed by the first row. Similarly, if a number of p coefficients in a column of matrix **B** share the same absolute value, $p-1$ multiplication operations can be skipped. The calculation of an entire column can be skipped if two columns of **B** are identical. Due to the nature of matrix multiplication, proper row and column swaps can be performed to make use of the above properties for skipping operations.

3.1.2 Computation & encoding-friendly structure

The structural unification loss $\mathcal{L}_u(\Phi)$ aims at pursuing a structure for learned weights that aligns with the GEMM process for computation acceleration. Also, we hope that the learned weights can be represented by as few bits as possible, so that they can be encoded effectively.

For each layer, weights W_i is normally a 4D tensor of size $(c_{in}, c_{out}, k_1, k_2)$, where c_{in} and c_{out} and the number of input and output channels respectively, and (k_1, k_2) gives the kernel size. The weight tensor can be reshaped as needed, resulting in equivalent matrix multiplciation with reshaped input and output tensors. Here, we reshape W_i into a 3D tensor of size $(c_{in}, c_{out}, k_1 \cdot k_2)$, and we reorder the $c_k = k_! \cdot k_2$ indices of W_i along the kernel axis, as shown in Fig. (2). The rationale is that a weight matrix in the (c_{in}, c_{out}) dimension should have locally correlated weights rather than being totally random in nature, because weight matrices combine input features to generate meaningful output features in an organized fashion.

We propose a locality-sensitive weight structure with a pairing locality-sensitive encoding scheme to make use of such local patterns for highly efficient weight compression. We divide the reshaped weight tensor W_i into 3D Coding Tree Units (CTU3D) as our basic processing units. The benefits of such a block-wise process mainly lie in two folds. First, blocking has been accepted as a general practice to speed up the GEMM computation in popular libraries like BLAS [6]. Second, encoding CTU3D individually avoids the inefficiency of generating a very large codebook as will be described in Sec. 4. By considering both aspects, we set our CTU3D to be $64 \times 64 \times c_k$ for balanced computation and encoding efficiency.

Note that when $k=1$, such as for 1×1 convolution, The CTU3D units degenerate to 2D matrices. Also, for boundary cases when c_{in} or c_{out} is not divisable by 64, the corresponding CTU3D along the boundaries will be smaller. In other words, $64 \times 64 \times c_k$ is the largest size for a CTU3D.

3.1.3 Structured unification

We further divide each CTU3D S_{ij} into $2 \times 2 \times 2$ blocks (2×2 for the degenerated 2D version), and weights within each block b_{ijl} are unified. Specifically, we set the weight coefficients in the block to have the same absolute value (*i.e.*, the mean of the absolute of the weight coefficients in b_{ijl}) while maintaining their original signs. We prefer such small blocks to prevent large prediction loss, because the larger the blocks, the more strict constraints we put on weight coefficients by enforcing them to share the same absolute values within blocks, and the less solution space we have in finding optimal weights for prediction.

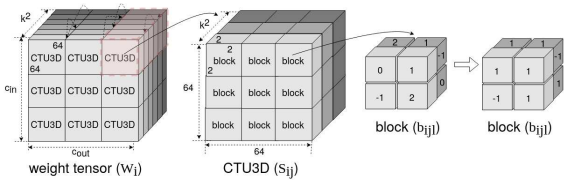


Figure 2: Illustration of the weight unification process. A general 4D weight tensor is firstly reshaped and reordered based on the CTU3D structure. Weights of a selected CTU3D unit is unified within partitioned blocks.

The potential loss introduced by such a unification operation can be measured by the standard deviation of the absolute of the weight coefficients:

$$\mathcal{L}_u(b_{ijl}) = \text{std}\left(|b_{ijl}|\right), \quad (4)$$

$$\mathcal{L}_u(S_{ij}) = \text{avg}_{b_{ijl} \in S_{ij}} \mathcal{L}_u(b_{ijl}). \quad (5)$$

Based on the above $\mathcal{L}_u(S_{ij})$, we search for the optimal way to reorder S_{ij} along the kernel axis by searching for $(k-1)/2$ pairs of indices in a greedy fashion. For each pair of indices (l_1, l_2), we first compute the 2D loss $\mathcal{L}_u(S_{ij})$ based on 2×2 blocks within each (c_{in}, c_{out}) plane along the kernel axis, and the first index l_1 is the one with the minimum loss. Then we exhaustively search for the optimal index l_2 to pair with l_1 with the 3D loss $\mathcal{L}_u(S_{ij})$ computed based on $2 \times 2 \times 2$ blocks using l_1 as the first index. Note that since c_k is normally an odd number, there will be a left-alone index for each CTU3D, and 2×2 blocks are used for for this index.

Then we rank all CTU3D units S_{ij} based on their optimal unification loss $\mathcal{L}_u(S_{ij})$ in accenting order. Given a ratio of unification q, weight coefficients of the top $q\%$ units are unified over their corresponding 2D or 3D blocks.

3.2. Iterative training

As mentioned before, an iterative training process is used to effectively find the optimal $\{W_i\}, \{M_i\}, \{U_i\}$ by alternating two steps:

Step 1: Given a unification ratio q, we conduct the above weight unification process described in Sec. 3.1 to obtain the unifier $\{U_i\}$, which computes a unified weight tensor for each W_i. This step actually minimizes the following unification loss $\mathcal{L}_u(\Phi)$ defined over the CTU3D structure:

$$\mathcal{L}_u(\Phi) = \sum\nolimits_i \sum\nolimits_j \mathcal{L}_u(S_{ij}) \quad (6)$$

Step 2: We fix the unified weight coefficients of $\{W_i\}$ unified by $\{U_i\}$, and conduct network fine-tuning to update

the remaining un-fixed weight coefficients of $\{W_i\}$ and the corresponding masks $\{M_i\}$ with normal back-propagation, based on the loss $\mathcal{L}_\Phi$ of Eqn. (3).

It is worth mentioning that in practice, the first weight unification step can automatically preserve the pruning structure in the mask M_i for most CTU3D units. This is because the pruned weights are set to zero already, resulting in very small unification loss of the corresponding units. Completely pruned out units will remain pruned out. Partially pruned out units will be reset to have a very small absolute value in most cases, which tends to be pruned out again in sucessive iterations.

As discussed before, such an iterative training process can effectively stabilize and speedup the learning process by separating and alternatively pursuing the two learning targets: structured weight unification for improved encoding and computation, and weight retraining for maintaining target prediction. Another benefit of such a separated process is that varying hyperparameters q and α can be used whose values can change during the training process to address on different targets at different times. For example, an increasing q as the iteration progresses results in a gradually increased amount of unified weights, and an increasing α results in a gradually increased amount of pruned weights.

4. 3D Pyramid-Based Encoding

In this seciton, we develop a 3D pyramid-based encoding method to make use of the CTU3D based unification weight structure for highly efficient weight compression.

The block-wise CTU structure has been used by the video coding society [17] for its balanced compression and computation performanc, where a video frame is usually partitioned into 64×64 CTUs as the basic units for encoding. Our $64\times64\times c_k$ CTU3D is an extension of the 2D CTU as the basic encoding unit to provide the computation and accuracy-preserving benefits for our weight unification. That is, each CTU3D is individually encoded by first generating an indiviudal codebook and then selecting the optimal encoding method using the codebook. Such a block-wise process can largely improve the encoding efficiency by avoiding the difficulty in finding an effective codebook for encoding the huge amount of weight coefficients all together. It also introduces the flexibility of using different coding schemes to encode different units for further improved compression.

4.1. Quantization

For each W_i, given a saturation value v_i and a bit depth d_i, a step size t_i can be computed as $t_i = v_i/(2^{d_i}-1)$ to divide the $(0, v_i)$ range into 2^{d_i} uniform bins. Each item w_{ij} in W_i is first truncated using v_i, and then quantized into a corresponding bin based on its absolute value $|w_{ij}|$. The original sign of w_{ij} is also kept, which will be represented by an additional bit. This gives a uniform quantization representation for all CTU3D $S_{ij} \in W_i$.

For each CTU3D S_{ij}, we can also compute a codebook using the newly developed palette coding approach in the HEVC extension of the screen content coding [17]. Each weight coefficient can be represented by the codeword index of the codebook, which gives an individual codebook-based quantization representation for each S_{ij}.

4.2. Entropy coding

Due to the sensitivity of the DNN prediction to the change of weight coefficients, lossless entropy coding is usually used to encode quantized weights. In this paper, we design 2 different pyramid-based coding schemes based on the 3D-Octtree structure. Also, we can choose to encode the codebook-based representation or the uniform quantization representation. Therefore, for each CTU3D S_{ij}, we have 4 different mix-and-match choices for entropy coding, and we select the optimal one based on the corresponding Rate-Distortion (RD) measurement: $RD = D + \gamma R$, where D is the MSE between the original and reconstructed signal and R is the bit count from entropy coding.

4.2.1 Pyramid-based coding

An octtree is a tree structure where each parent node has 8 children. A 3D-Octtree partitions a 3D tensor by recursively subdividing it along the 3 axes into eight octants.

Method 1: 3D-Unitree: For non-leaf nodes, if all children of a node share the same absolute unified value (or codebook index), the node is assigned value 0. Otherwise the node is assigned value 1. For leaf nodes, if all nodes of the same parent node have the same absolute unified value (or codebook index), these nodes are assigned value 0. Otherwise, value 1 is assigned to these nodes. As illustrated in Fig. 3, we can skip encoding nodes by taking advantage of the unified weight structure, where only the absolute unified value is encoded for skpped nodes.

Method 2: 3D-Tagtree: Each non-leaf node takes the maximum absolute value (or codebook index) of its children, and the leaf node takes its original absolute value (or codebook index). As illustrated in Fig. 3, we can skip encoding nodes by taking advantage of the pruned weight structure obtained from our retraining process, since nodes in a pruned branch share the same value 0 and can be skipped.

5. Experiments

We evaluate the proposed method on a set of representative DNN benchmarks: ResNet-50 [10], MobileNet-V2 [22] for ImageNet image classification; Autoencoder for image compression over CIFAR-100; and MLP for audio sound event classification [1]. These benchmarks cover different tasks including image and audio classification and

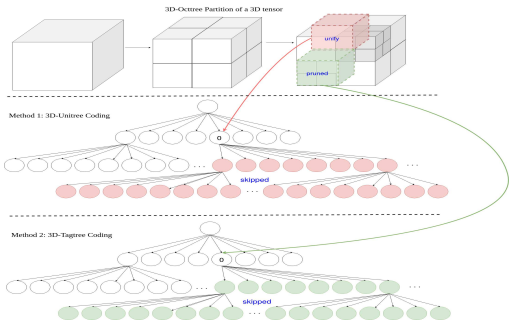


Figure 3: Illustration of pyramid-based coding, which benefits from the unification structure for efficient encoding.

end-to-end image compression, and the size of the tested network models vary dramatically, ranging from the moderately large ResNet-50 to the relatively condense MobileNet-V2 and to a simple 2-layer MLP. These benchmarks (models and corresponding datasets) are also selected as representative test cases by the MPEG Neural Network Compression Standard group for required core experiments [19]. More details about the evaluation setup can be found there.

5.1. Implementation details

To clearly show the benefit of our algorithm, we evaluate two different versions of our method. First, we set $\alpha = 0$ and conduct weight unification and retraining without weight pruning. That is, starting from the pre-trained model using the corresponding benchmark training data, we evaluate the ability of our algorithm in compressing the pre-trained model and accelerating the inference computation, without the component of weight pruning. Then, we set $\alpha > 0$, and conduct the whole weight unification and pruning process described in Sec. 3.2. In this case, the unstructured weight pruning method of [2] is used. This weight pruning method is actually adopted by the MPEG Neural Network Compression Standard group as one of the parameter pruning methods [20]. We follow the experiment setup in [19], where α is automatically determined by presetting a ratio m between the validation regularization loss and the validation data loss. That is, we use the reported hyperparameter m to automatically calculate α following [19], and then fix this α throughout our retraining process.

For our weight unification step, we skip the first input layer and the last output layer. This is because these two layers are very sensitive to parameter changes, which may lead to significant prediction loss. Also, these two layers usually do not have a large amount of parameters due to the small number of input channels and the small number of output targets (*e.g.*, number of classes). Our algorithm implementation is based on PyTorch, and experiments are conducted using the NVIDIA DGX station with 4 Tesla V100 GPUs.

5.2. ImageNet classification

Table 1 shows the performance of the two versions of our algorithm with $\alpha = 0$ and $\alpha > 0$ over the ImageNet classification task using ResNet-50 and MobileNet-V2. The Top-1 accuracy and Top-5 accuracy are used to measure the prediction performance. The complexity of inference computation is measured by MACs (about $0.5 \times$ FLOPs). For all the experiments, we set the maximum tolerable prediction performance drop to be 2% and report results of different unification ratios q. The results of $q = 0$ correspond to the original pre-trained model when $\alpha = 0$, or the unstructurally pruned model by [2] when $\alpha > 0$. Note that the unstructually pruned input model ($\alpha > 0$, $q = 0$) can not improve the inference computation in general, and will have the same number of MACs with the original pre-trained model.

To better understand the performance of our system, we list the prediction performance of the unified model after the two-step retraining and before the following quantization and entropy coding, named as "u-top-1", "u-top-5". We also list the final prediction performance of our system after the further encoding process, named as "c-top-1", "c-top-5". The compression rate "$c-rate$" of the original model versus the final encoded model measures the compression performance of our system. The lossy encoding process will cause performance drop, and an optimal model can be empirically determined based on the validation set by examining the RD measurements.

q	u-top-1	u-top-5	c-top-1	c-top-5	c-rate	g-macs
0	76.15	92.87	75.10	92.38	10.56	4.11
10	76.30	92.94	75.07	92.28	10.63	3.83
20	76.23	92.89	74.66	92.15	10.72	3.55
30	75.77	92.64	74.72	92.31	10.07	3.27

(a) ResNet-50 ($\alpha = 0$)

q	u-top-1	u-top-5	c-top-1	c-top-5	c-rate	g-macs
0	69.00	88.70	68.6	88.26	8.9	4.11
10	69.57	89.05	68.57	88.40	14.5	3.87
20	68.83	88.62	68.17	87.99	13.16	3.56
30	68.30	88.24	66.67	87.11	9.02	3.27

(b) ResNet-50 ($\alpha > 0$)

q	u-top-1	u-top-5	c-top-1	c-top-5	c-rate	g-macs
0	71.87	90.29	70.41	89.71	6.85	0.32
10	71.60	90.19	70.29	89.42	7.02	0.3
20	71.45	90.07	70.34	89.29	7.2	0.28
30	71.05	89.85	69.62	88.95	7.44	0.26
40	70.42	89.63	69.60	89.17	6.59	0.24
50	70.25	89.44	69.38	88.83	6.87	0.21
60	69.26	88.97	68.71	88.66	7.2	0.19

(c) MobileNet-V2 ($\alpha = 0$)

q	u-top-1	u-top-5	c-top-1	c-top-5	c-rate	g-macs
0	65.06	86.41	64.65	86.15	7.06	0.32
10	67.62	88.19	66.38	87.42	9.25	0.3
20	66.36	87.19	65.14	86.52	9.73	0.28
30	64.06	86.03	64.50	84.18	10.38	0.26

(d) MobileNet-V2 ($\alpha > 0$)

Table 1: Performance of unified and encoded model for ImageNet classification with different q. g-macs denotes gigaMACs

From the table, we can see that for ImageNet classification, within only 2% performance drop, we can ac-

celerate the inference of ResNet-50 by 0.83 gigaGMACs with about $10\times$ compression rate, for both pruned or original pre-trained model. As for the already highly efficient MobileNet-V2, we can still accelerate its inference by 0.11 or 0.06 gigaMACs and compress the model by about $7\times$ or $9\times$ for the original pre-trained model or the pruned model. Actually, when the unification ratio is small, *e.g.*, $q = 10\%$, our unification retraining actually improves the prediction performance from the original input models most of the time. Such promising results indicate that there are indeed redundancy in the original pre-trained or already pruned model. Our approach can reduce such redundancy by using the unification requirement as a regularizer and learn a better model for both prediction and computation.

5.3. Audio classification and image compression

The network for both audio classification and image compression are quite small, and we observe unbalanced performances of our structured weight unification process and the final encoding process over these tasks. Specifically, Table 2 shows the performance of the unified model for audio classification, where we improve the prediction performance on top of the original input models most of the time, and also speed up the original computation by $2\times$ (*i.e.*, 53.13 millionMACs). Similarly, Table 3 shows the performance of the image compression task, where PSNR and SSIM are used as measurements. Again, we consistently improve the prediction performance over the original input models, and speed up the computation by 27.16 millionMACs, which is roughly $3\times$.

q	u-top-1	u-top-5	m-macs
0	58.27	91.85	113.74
10	60.61	91.23	107.84
20	60.74	91.36	101.94
30	60.49	91.36	96.03
40	60.62	91.23	90.13
50	59.88	91.48	84.22
60	60.00	91.48	78.32
70	60.74	91.48	72.42
80	60.98	91.60	66.51
90	58.64	90.86	60.61

(a) $\alpha = 0$

q	u-top-1	u-top-5	m-macs
0	60.37	92.96	113.74
10	60.99	92.72	107.84
20	61.23	92.72	101.94
30	60.49	92.47	96.03
40	59.75	91.85	90.13
50	60.49	91.98	84.22
60	59.75	91.72	78.32
70	59.88	92.59	72.42
80	60.62	92.10	66.51
90	60.86	90.37	60.61

(b) $\alpha > 0$

Table 2: Performance of unified model for audio classification with different q. m-macs denotes millionMACs

q	u-psnr	u-ssim	m-macs
0	29.79	0.956	46.05
10	29.87	0.957	43.04
20	29.86	0.957	40.03
30	29.91	0.957	37.00
40	29.88	0.957	33.99
50	29.87	0.957	30.95
60	29.85	0.957	27.95
70	29.89	0.956	24.93
80	29.88	0.957	21.91
90	29.92	0.957	18.89

(a) $\alpha = 0$

q	u-psnr	u-ssim	m-macs
0	29.91	0.958	46.05
10	29.97	0.958	43.04
20	29.99	0.958	40.03
30	29.94	0.959	37.00
40	30.05	0.959	33.99
50	29.98	0.958	30.95
60	29.93	0.959	27.95
70	29.96	0.958	24.93
80	29.97	0.958	21.91
90	30.01	0.959	18.89

(b) $\alpha > 0$

Table 3: Performance of unified model for image compression with different q. m-macs denotes millionMACs

Comparing with the structurally unified model, the encoding process causes additional prediction performance drop. Here we set the maximum tolerable prediction performance drop to be 3%, and Table 4 shows the results within this range. We can see that our method still can reduce the already small model size by $5 \sim 6\times$ and $10 \sim 20\times$ for image compression and audio classification tasks, respectively, and at the same time, improve the inference computation by $3 \sim 6$ millionMACs and $17 \sim 23$ millionMACs.

q	u-top-1	u-top-5	c-top-1	c-top-5	c-rate	m-macs
0	58.27	91.85	58.01	90.86	7.02	113.74
10	60.61	91.23	59.26	91.48	9.24	107.84
20	60.74	91.36	58.15	90.37	13.11	101.94
30	60.49	91.36	55.68	90.37	10.35	96.03

(a) audio classification $\alpha = 0$

q	u-top-1	u-top-5	c-top-1	c-top-5	c-rate	m-macs
0	60.37	92.96	59.80	92.01	7.11	113.74
10	60.99	92.72	60.62	92.84	16.32	107.84
20	61.23	92.72	59.63	92.59	17.91	101.94
30	60.49	92.47	59.26	92.59	19.24	96.03
40	60.49	92.47	57.16	92.10	21.03	90.13

(b) audio classification $\alpha > 0$

q	u-psnr	u-ssim	c-psnr	c-ssim	c-rate	m-macs
0	29.79	0.956	28.07	0.93	3.20	46.05
10	29.87	0.957	26.46	0.91	5.36	43.04

(c) image compression $\alpha = 0$

q	u-psnr	u-ssim	c-psnr	c-ssim	c-rate	m-macs
0	29.91	0.958	29.20	0.95	3.62	46.05
10	29.97	0.958	28.88	0.95	5.76	43.04
20	29.99	0.958	28.19	0.94	6.15 40.03	

(d) image compression $\alpha > 0$

Table 4: Performance of unified and encoded model for audio classification and image compression within 2% prediction drop

6. Conclusion

We propose a structured joint weight unification and weight encoding framework for reducing the model size and speeding up the inference computation of DNNs. Our structured weight unification method unifies weights according to hardware-friendly structures to facilitate the following encoding as well as accelerating the underlying convolution operation. Our locality-sensitive encoding method is designed to make use of the unified weight structures to effectively compress the DNN models. Compared with traditional unstructured weight pruning, our structured approach can largely reduce inference computation. Compared with traditional structured weight pruning, our method unifies weight structures instead of removing them, which better preserves the network capacity to avoid the prediction performance degradation. Our generic two-step training framework can accommodate different encoding algorithms, different weight unification structures, and different DNN architectures. Experiments over several benchmarks show the effectiveness of our method. One future work is to further investigate improved encoding methods or joint retraining process to alleviate the unbalanced performance of unified and encoded models for small-size models.

References

[1] S. Adavanne, A. Politis, and T. Vertanen. A multi-room reverberant dataset for sound evnet localization and detection. *Detection and Classification of Acoustic Scenes and Events 2019 Workshop (DCASE2019)*, 2019. 6

[2] C. Aytekin, F. Cricri, T. Wang, and E. Aksu. Response to the call for proposals on neural network compression: Training highly compressible neural networks. *ISO/IEC JTC1/SC29/WG11 m47379*, 2019. 7

[3] Y. Cheng, D. Wang, P. Zhou, and T. Zhang. A survey of model compression and acceleration for deep neural networks. *IEEE Signal Processing Magazine, Special Issue on Deep Learning for Image Understanding*, 2019. 3

[4] S. Chetlur, C. Woolley, P. Vandermersc, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*, 2014. 3

[5] X. Dong, S. Chen, and S. Pan. Learning to prune deep neural networks via layer-wise optimal brain surgeon. *NIPS*, pages 4857–4867, 2017. 1, 3

[6] R. Geijn and J. Huang. How to optimize gemm. 5

[7] Y. Guo, A. Yao, and Y. Chen. Dynamic network surgery for efficient dnns. *NIPS*, pages 1379–1387, 2016. 1, 3

[8] S. Han, H. Mao, and W. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *ICLR*, 2016. 1, 3

[9] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. *NIPS*, pages 1135–1143, 2015. 1, 3

[10] K. He, X. Ren, and J. Sun. Deep residual learning for image recognition. *CVPR*, pages 770–778, 2016. 6

[11] Y. He, X. Zhang, and J. Sun. Channel pruning for accelerating very deep neural networks. *ICCV*, 2017. 1, 3

[12] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *NIPS Deep Learning and Representation Learning Workshop*, 2015. 1

[13] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions. *BMVC*, 2014. 1

[14] K.Ullrich, E. Meeds, and M. Welling. soft weight-sharing for neural network compression. *CoRR*, 2017. 3

[15] V. Lebedev and V. Lempitsky. Fast convnets using group-wise brain damage. *CVPR*, pages 2074–2082, 2016. 1

[16] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. Graf. Pruning filters for efficient convnets. *ICLR*, 2017. 1, 3

[17] S. Liu, X. Xu, S. Lei, and K. Jou. Overview of hevc extensions on screen content coding. *SIP Industrial Technology Advances*, 4:1–12, 2015. 6

[18] Document N18162. Updated call for proposals on neural network compression. *ISO/IEC JTC 1/SC 29/WG 11*, 1 2019. 1, 2

[19] Document N18782. Description of core experiments on compression of neural networks for multimeida content description and analysis. *ISO/IEC JTC 1/SC 29/WG 11*, 10 2019. 2, 7

[20] Document N18785. Test model 2 of compression of neural networks for multimedia content description and analysis. *ISO/IEC JTC 1/SC 29/WG 11*, 10 2019. 7

[21] A. Ren, J. Li, T. Zhang, S. Ye, W. Xu, X. Qian, X. Lin, and Y. Wang. Admm-nn: An algorithm-hardware co-design framework of dnns using alternating direction methods of multipli-erss. *International conference on Architectural Support for Programming Languages and Operating Systems*, 2019. 1, 2, 3, 4

[22] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. *CVPR*, pages 4510–4520, 2018. 1, 6

[23] G. Schindler, W. Roth, F. Pernkopf, and H. Froning. Parameterized structured pruning for deep neural networks. *arXiv preprint: https://arxiv.org/pdf/1906.05180.pdf*, 2019. 1

[24] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning structured sparsity in deep neural networks. *NIPS*, pages 2074–2082, 2016. 1, 3

[25] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng. Quantized convolutional neural networks for mobile devices. *CVPR*, 2016. 1, 3

[26] S. Wu, G. Li, F. Chen, and L. Shi. Training and inference with integers in deep neural networks. *ICLR*, 2018. 1, 3

[27] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen. Incremental network quantization: Towards lossless cnns with low-precision weights. *ICLR*, 2017. 1, 3

[28] H. Zhou, J. Alvarez, and F. Porikli. Less is more: Towards compact cnns. *ECCV*, 2016. 1, 3