Hybrid Network Compression via Meta-Learning

Jianming Ye
jmye@pku.edu.cn
Peking University
Beijing, China

Shiliang Zhang
slzhang.jdl@pku.edu.cn
Peking University
Beijing, China

Jingdong Wang
jingdw@microsoft.com
Microsoft Research Asia
Beijing, China

ABSTRACT

Neural network pruning and quantization are two major lines of network compression. This raises a natural question that whether we can find the optimal compression by considering multiple network compression criteria in a unified framework. This paper incorporates two criteria and seeks layer-wise compression by leveraging the meta-learning framework. A regularization loss is applied to unify the constraint of input and output channel numbers, bit-width of network activations and weights, so that the compressed network can satisfy a given Bit-OPerations counts (BOPs) constraint. We further propose an iterative compression constraint for optimizing the compression procedure, which effectively achieves a high compression rate and maintains the original network performance. Extensive experiments on various networks and vision tasks show that the proposed method yields better performance and compression rates than recent methods. For instance, our method achieves better image classification accuracy and compactness than the recent DJPQ. It achieves similar performance with the recent DHP in image super-resolution, meanwhile saves about 50% computation.

CCS CONCEPTS

• **Computing methodologies** → **Computer vision**.

KEYWORDS

Neural Networks, Pruning, Quantization, Neural Network Compression

ACM Reference Format:
Jianming Ye, Shiliang Zhang, and Jingdong Wang. 2021. Hybrid Network Compression via Meta-Learning. In *Proceedings of the 29th ACM Int'l Conference on Multimedia (MM '21), Oct. 20–24, 2021, Virtual Event, China.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3474085.3475353

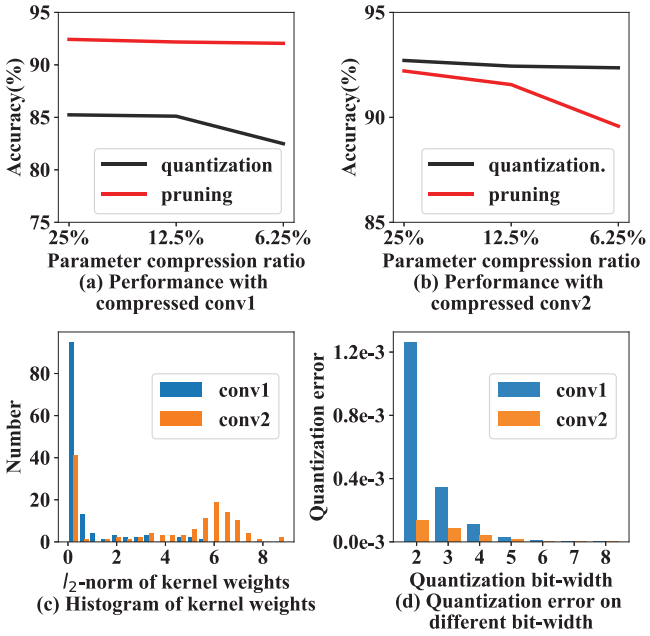


Figure 1: (a) and (b) compare the CNN performances after applying pruning and quantization on the conv1 and conv2 layers of VGG-small, respectively. (c) and (d) show the l_2-norm distribution of kernel weights and quantization errors of conv1 and conv2 layers. It is clear that, conv1 exhibits smaller l_2-norm of kernel weights, thus is more suitable to be compressed by pruning. Conv2 maintains lower quantization errors, thus is more suitable for quantization. Observations in (c) and (d) are consistent with the conclusions in (a) and (b).

1 INTRODUCTION

Many milestone works [11, 31] on deep Convolutional Neural Network (CNN) architectures have significantly boosted the performance of various computer vision tasks. The million-scale parameters and billion-scale Floating Point Operations (FLOPs) consumed by deep CNNs make it important to study CNN compression strategies, with the goal of decreasing the memory and computations costs and maintaining the high performance of original CNNs.

There are two major lines of CNN compressions algorithms. Both of them have to resort to certain criteria to find redundancies in floating-point CNNs. Neural network pruning targets to identify and delete unimportant connections in the network [12, 14]. Quantization methods replace floating-point parameters with low bit-wise ones to reduce time complexity and memory cost [8, 26, 43]. Since most of the existing works only consider one criterion, this raises a natural question that how one compression method might take account of multiple criteria in finding the optimal compression, *e.g.*, setting high pruning rates if lots of redundant connections exist or using low bit-wise quantization if the variance of parameters is small.

MM '21, October 20–24, 2021, Virtual Event, China.

ACM ISBN 978-1-4503-8651-7/21/10…$15.00
https://doi.org/10.1145/3474085.3475353

Our experiments reveal that different layers show different adaptations for pruning and quantization, *i.e.*, some layers could maintain the performance after pruning most of the channels, while some others can be quantized with low bit-rate precision as shown in Fig. 1 (a) and (b), respectively. Fig. 1 (c) and (d) show the l_2-norm distribution of kernel weights and the quantization error under different bit-width. According to [12], channels with low l_2-norm values are less important and could be removed by pruning. Therefore, conv1 is more suitable to be compressed by pruning than conv2. It also can be observed that, conv2 maintains lower quantization errors, thus is more suitable for quantization than conv1. Different compression methods on conv1 and conv2 bring different performance degradations.

This paper is thus inspired to jointly consider two criteria to seek an optimal layer-wise compression for CNNs. How to choose a suitable combination of different compression criteria for each CNN layer is a challenge and has not been extensively studied. Some recent works like [19, 40] leverage unstructured pruning and quantization in a joint framework. However, unstructured pruning leads to irregular weight parameters, which are not favorable for computation acceleration in real applications on hardware. To seek more effective compression strategies, our method focuses on structured pruning and quantization. A simple way to combine pruning and quantization is to directly combine the two methods in a unified framework like [38]. It simply applies two independent pruning and quantization methods trained together for a network, and empirically sets the strength of quantization and pruning. Therefore, it still lacks an adaptive mechanism to balance the effects of pruning and quantization on different layers.

To seek a more reasonable combination of those two criteria, this paper proposes an end-to-end framework to adaptively learn layer-wise parameters for pruning and quantization. We divide the training procedure into the searching stage and the fine-tuning stage. Inspired by DHP [22] and LW-DNA [23], the searching stage uses the latent vectors attached to each convolutional layer to guide the layer pruning. It also leverages quantization parameters for each convolutional layer to guide the quantization. This is achieved by a hypernetwork, which leverages the latent vectors of the current layer to control the output channel number, and uses latent vectors of the previous layer to control the input channel number. By passing latent vectors and the quantization parameters through the hypernetwork, we can get its outputs, which are used as the pruned and quantized convolutional kernels. The subsequent fine-tuning stage discards the hypernetwork and keeps the network structures, *i.e.*, the channels and quantization parameters are fixed, the other network parameters are fine-tuned to get a high-performance network.

We conduct experiments using different network architectures on image classification and single image super-resolution tasks, respectively. Experimental results show that, our method achieves a substantially higher compression rate than recent works, while maintains comparable or better performance. For instance, our method achieves higher classification accuracy and compactness than the recent DJPQ [38]. It gets similar performance with the recent DHP [22] on image super-resolution, meanwhile saves about 50% computation. We hence conclude that, considering multiple criteria leads to a more effective compression strategy. To the best of our knowledge, this is an original work jointly considering network pruning and quantization in an end-to-end meta learning framework. Besides this novel framework, this paper also introduces an iterative compression constraint for optimizing the compression procedure. Those two components effectively guarantee the promising performance of proposed methods.

2 RELATED WORK

Most of the existing works consider only one compression criterion, *i.e.*, either network pruning or quantization. A few recent works focus on the combination of two criteria. This section briefly reviews those three categories of methods: structured pruning, network quantization, and joint optimization of pruning and quantization.

Structured pruning methods can be divided into two categories according to their computed pruning parameters at each layer, *i.e.*, fixed pruning and layer-wise pruning, respectively. Fixed pruning methods leverage different criteria to determine the importance of each channel, and then delete the unimportant channels. For example, [15] proposed a two-step channel pruning method. It first selects unimportant channels based on Lasso regression values. Then it prunes to reach a given target pruning ratio, and then fine-tunes weights. [12, 14] use l_2-norm and geometric median of the filters as the criteria to determine the channel importance, respectively. The approach proposed in [28] leverages first-order Taylor expansion for the loss function. Many works leverage second-order Taylor expansion to explore inter-channel dependency, such as [29, 33]. Adaptive pruning methods use learning methods to choose suitable pruning ratios for each layer. For example, [13] uses reinforcement learning to decide the pruning ratios for different layers. Another method [25] trains a hypernetwork to predict the performance of each pruned sub-net, and finally obtains a pruned network with good performance. The pruning approach in [4] leverages the concept of variational information bottleneck (VIB) proposed in [34]. By adding gates to the network and employing a suitable regularization term, [4] manages to determine the importance of each channel and prune the network with high compression ratio. [22] and [23] proposed a novel hypernetwork to learning the network parameters with latent vectors, which is able to determine the channel number in each layer with end-to-end training.

Network quantization methods can reduce the time complexity and memory complexity by quantizing the parameters into low bit-width ones. Most works use fixed-bit quantization. For example, [2, 8, 26] use fixed 4 or 8-bit quantization to compress the network. Some methods like [20] proposed binary quantization for weights and activations, but this method achieves low performance even with activations unquantized. With mixed-precision opponents, [35] achieves higher compression ratio and competitive performance. Many others determine the bit-width for each layer adaptively. For example, DQ [35] leverages straight-through estimator (STE) and defines a continuous relaxation to learn the bit-width of both weights and activations by gradient back-propagation. Some other works like [6] uses reinforcement learning to learn the bit-width of each layer. In [36], the total variance of each layer is used to determine the bit-width adaptively.

Jointly optimize pruning and quantization has the potential to achieve a better compression ratio. However, few works have

been proposed in this category. An early work [9] considers pruning and quantization in separated steps for network compression. Some works [19, 40] are proposed to jointly optimize unstructured pruning and quantization. However, it is hard to implement unstructured pruning with typical hardware. A recent work DJPQ [38] is proposed combining pruning method VIB-net [4, 34] and quantization method DQ [35]. Our work shares certain similarity with DJPQ [38] in that, it also leverages DQ [35] for end-to-end training.

Differently, instead of simply combining pruning and quantization in a framework like DJPQ, our method adaptively conducts those two criteria for each layer. Also, our end-to-end training is implemented with a meta-learning framework. DJPQ [38] leverages pruning loss and regularization loss to guide the pruning process and quantization process. Our method manages to unify these two compression processes into the meta-learning framework. What's more, our method conducts the compression process with a single regularization loss. Thanks to the unified framework and training scheme, our method manages to adaptively compress each layer in a network. As shown in the experiments, our method outperforms DJPQ in aspects of image classification accuracy and network compression ratio.

3 FORMULATION

Given a CNN with N-layers and the corresponding convolutional kernels $\mathbb{W} = \{\mathbf{W}_n\}_{n=1:N}$, as well as a training set composed of an image set X and its label set Y. Our goal is to train a compressed CNN, which satisfies the given Bit-OPerations counts (BOPs) [38] upper bound, meanwhile preserves the performance of the original CNN.

The original CNN can be compressed by either applying pruning or quantization to floating-point kernels at each layer. For a n-th layer with a c_n-channel convolutional kernel $\mathbf{W}_n$, the pruning can be defined by a c_n-dim vector α_n, which indicates the importance of each channel. The pruning can be computed as,

$$\mathbf{W}_n^{(P)} = \mathbf{W}_n[\mathbf{1}(\alpha_n[1:c_n] > t)], \tag{1}$$

where $\mathbf{W}_n^{(P)}$ denotes parameters after pruning. t is the pruning threshold and $\mathbf{1}(\cdot)$ is an indicator function. We set t as $5e-3$ following [22].

Meanwhile, the kernel $\mathbf{W}_n$ can be compressed by quantization defined by the parameter $\Theta_n = \{m_n^{conv}, d_n^{conv}, \text{bit}_n^{conv}\}$, which includes the quantization value range m_n^{conv}, step size d_n^{conv}, and bit-width bit_n^{conv} of the quantization. With Θ_n the quantization can be denoted as (we omit ^{conv} to simply the description),

$$w_n^{(Q)} = \begin{cases} \mathcal{S}(w_n) \cdot d_n \cdot \lfloor \dfrac{|w_n|}{d_n} + \dfrac{1}{2} \rfloor, |w_n| \le m_n \\ \mathcal{S}(w_n) \cdot m_n \qquad , |w_n| > m_n \end{cases} \tag{2}$$

$$m_n = (2^{\text{bit}_n - 1}) \cdot d_n, \tag{3}$$

where the scalar $w_n^{(Q)}$ denotes one of quantized parameters in $\mathbf{W}_n^{(Q)}$. $\mathcal{S}(\cdot)$ is the sign function. Similarly, the activation $\mathbf{O}_n$ of the n-th layer can be quantized by parameter $\Omega_n = \{m_n^{act}, d_n^{act}, \text{bit}_n^{act}\}$ containing the quantization value range m_n^{act}, step size d_n^{act}, and bit-width bit_n^{act}, with similar strategy as shown in Eq. (2).

Applying both pruning with parameters α_n and quantization with parameters Θ_n lead to compressed parameters $\mathbf{W}_n^{(PQ)}$. The resulting computation complexity of n-th layer can be computed by counting the BOPs as,

$$\text{BOPs}_n = \text{MACs}_n \times \text{bit}_n^{conv} \times \text{bit}_{n-1}^{act}, \tag{4}$$

where bit_n^{conv} and bit_{n-1}^{act} denote bit-width of parameters in n-th layer and activations from the $(n-1)$-th layer respectively. MACs_n counts the Multiply-And-Accumulate operations (MACs) number in the n-th layer. It is computed as,

$$\text{MACs}_n = c_{n-1} \times c_n \times \text{size}_{n-1}^{act} \times \text{size}_n^{conv}, \tag{5}$$

where c_{n-1} and c_n denote the input and output channel number respectively. size_n^{conv} and size_{n-1}^{act} denote the kernel size in the n-th layer and spacial size of activations from the previous layer respectively.

As illustrated in Fig. 1, different layers in the CNN are suited for different combinations of pruning and quantization. This paper is motivated to seek optimal α_n, Θ_n and Ω_{n-1} for each layer. We denote the collection of parameters for pruning and quantization as $\mathbb{A} = \{\alpha_n\}_{n=1:N}$, $\mathbb{B} = \{\Theta_n\}_{n=1:N}$ and $\mathbb{C} = \{\Omega_n\}_{n=1:N-1}$. Our training is expected to seek optimal $\mathbb{A}$, $\mathbb{B}$ and $\mathbb{C}$ to make the computation cost below a given upper bound, meanwhile minimize the errors computed on Y, *i.e.*,

$$\mathcal{L}_t = \arg\min_{\mathbb{A},\mathbb{B},\mathbb{C},\mathbb{W}} \sum \text{E}(\text{CNN}(\mathbb{W}, X), Y), \quad \text{s.t.} \sum_{n=1:N} O_n < R \cdot \hat{O} \tag{6}$$

where $\text{E}(\cdot)$ is the loss function computed with the CNN prediction and ground truth label Y. $\mathcal{L}_t$ is the task specific loss, like classification loss for classification tasks and L_1 loss for single image super-resolution task. R is the target compression rate, and $\hat{O}$ is the BOPs of the original CNN without compression.

To optimize pruning and quantization in a joint framework, we leverage meta-learning to adaptively seek α_n, Θ_n, Ω_{n-1} as well as $\mathbf{W}_n^{(PQ)}$ for each layer. In other words, for each convolutional layer, we use a hypernetwork to learn the convolutional kernels. The hypernetwork learns latent vectors for both the previous layer and current layer, which helps to infer pruning parameters for the current layer. Also, quantization parameters of activations from previous layer Ω_{n-1} and quantized weights from current layer Θ_n are learned in the hypernetwork to help learning pruned and quantized weights and quantized activations. The computation of hyernetwork for the n-th layer can be formulated as,

$$\mathbf{W}_n^{(PQ)}, \mathbf{O}_{n-1}^{(Q)} = \mathcal{H}_n(\mathbf{O}_{n-1}, \mathbf{H}_n, \alpha_n, \alpha_{n-1}, \Theta_n, \Omega_{n-1}), \tag{7}$$

where $\mathcal{H}_n$ is the hypernetwork, and $\mathbf{H}_n$ denotes the convolutional parameters of the hypernetwork to be optimized. $\mathbf{O}_{n-1}$ denotes the output from the previous layer and $\mathbf{O}_{n-1}^{(Q)}$ denotes the quantized activation.

With quantized kernels and activations from the previous layer, the output of the n-th layer can be calculated by convolution operation,

$$\mathbf{O}_n = \mathbf{W}_n^{(PQ)} \otimes \mathbf{O}_{n-1}^{(Q)}, \tag{8}$$

where $\mathbf{O}_n$ can be propagated to the output layer to compute the task specific loss with $\text{E}(\cdot)$.

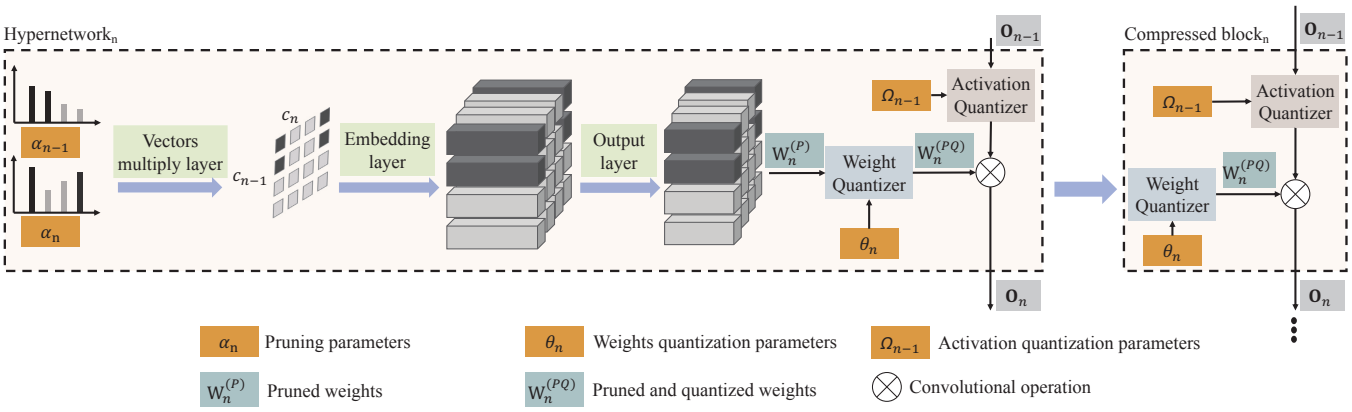


Figure 2: Illustration of our pipeline for joint pruning and quantization in searching stage and fine-tuning stage. α_n and α_{n-1} are the pruning parameters, e.g., channels with smaller values in α_n and α_{n-1} are pruned. The hypernetwork$_n$ learns Θ_n and Ω_{n-1} to conduct the quantization for convolutional kernels and activations. After training, we use the quantization parameters Θ_n, Ω_{n-1} and the pruned and quantized weights $W_n^{(PQ)}$ for fine-tuning stage training.

To jointly optimize pruning and quantization, we propose a network structure in Fig. 2. The training of the proposed network is divided into two processes, *i.e.* the searching stage and the fine-tuning stage. In the searching stage, we update the pruning parameters and quantization parameters using the designed hypernetwork. Then, it predicts the quantized convolutional kernels for the backbone network. During training, a regularization loss $\mathcal{L}_{REG}$ for network computation complexity is applied. Thus, we define the final loss $\mathcal{L}$ as:

$$\mathcal{L} = \mathcal{L}_t + \lambda \mathcal{L}_{REG}, \tag{9}$$

where the λ is the regularization loss weight. $\mathcal{L}_{REG}$ can be found from [38]. It is designed to optimize pruning parameters and quantization parameters to satisfy a certain BOPs constraint. Following [22], we apply l_1 sparsity on latent vectors. The network structure is fixed after the searching stage. In the fine-tuning stage, the hypernetworks are discarded, and the compressed network is fine-tuned with task specific loss. The following parts present the detailed implementation of our method.

4 IMPLEMENTATIONS

Different from traditional quantized convolutional networks, the weights and quantization parameters of our convolutional layers are learned adaptively by proposed hypernetworks. Referring to [22], we design our hypernetwork structure as shown in Fig. 2. For each layer, the hypernetwork first generates a pruned convolutional kernel, then quantizes it as the final output. The following parts present the implementation of pruning and quantization, respectively.

4.1 Implementation of Pruning

As in Sec. 3, we denote the pruning parameters from the previous layer and current layer as α_{n-1} and α_n respectively. α_{n-1} and α_n represent the weight of each channel in the convolutional kernel of the n-th layer. By training with l_1 constrain on pruning parameters, weights for unimportant channels in latent vectors will be reduced, which results in channel pruning. More specifically, hypernetwork implements the pruning by firstly computing a latent matrix from α_{n-1} and α_n, namely,

$$\mathbf{M}_n = \alpha_{n-1}^T \cdot \alpha_n + \mathbf{B}_{n_0}, \tag{10}$$

where both $\mathbf{M}_n$ and $\mathbf{B}_{n_0}$ have the size of $c_{n-1} \times c_n$. $\mathbf{B}_{n_0}$ is a trainable bias parameter. As illustrated in Fig. 2, $\mathbf{M}_n$ is passed through two point-wise convolutional layers to compute a pruned convolutional kernel. We denote the computation as,

$$\mathbf{W}_n^{(P)} = (\mathbf{M}_n \otimes \mathbf{H}_{n_1} + \mathbf{B}_{n_1}) \otimes \mathbf{H}_{n_2} + \mathbf{B}_{n_2}, \tag{11}$$

where $\mathbf{H}_{n_1}$ and $\mathbf{H}_{n_2}$ are convolutional kernels in the hypernetwork. $\mathbf{B}_{n_1}$ and $\mathbf{B}_{n_1}$ are corresponding bias parameters. Parameters of the hypernetwork, *i.e.*, $\{\alpha_n, \mathbf{B}_{n_0}, \mathbf{B}_{n_1}, \mathbf{B}_{n_2}, \mathbf{H}_{n_1}, \mathbf{H}_{n_2}\}$ are learned through end-to-end training with loss function defined by Eq. (9).

We set the size of $\mathbf{H}_{n_1}$ as $c_{n-1} \times c_n \times l \times 1$. Also, we set the size of $\mathbf{H}_{n_2}$ as $c_{n-1} \times c_n \times l \times \mathrm{size}_n^{conv}$. l is the output channel size of the embedding layer. $\mathbf{B}_{n_1}$ and $\mathbf{B}_{n_2}$ thus have the size of $c_{n-1} \times c_n \times l$ and $c_{n-1} \times c_n \times \mathrm{size}_n^{conv}$, respectively. The generated convolutional kernel $\mathbf{W}_n^{(P)}$ has the size of $c_{n-1} \times c_n \times \mathrm{size}_n^{conv}$.

4.2 Implementation of Differentiable Quantization

Θ_n and Ω_{n-1} are hence adopted by the hypernetwork to quantize $\mathbf{W}_n^{(P)}$ and the activation $\mathbf{O}_{n-1}$, respectively. We proceed to introduce a differentiable quantization to optimize Θ_n and Ω_{n-1} through end-to-end learning. The following part takes the quantization from $\mathbf{W}_n^{(P)}$ to $\mathbf{W}_n^{(PQ)}$ as an example. The quantization to $\mathbf{O}_{n-1}$ can be implemented in similar way.

As shown in Eq. (3), the value of bit-width bit_n^{conv} can be inferred from the step size d_n^{conv} and quantization value range m_n^{conv}. By training m_n^{conv} and d_n^{conv} with gradient back-propagation, we can update b_n^{conv} with Eq. (3). The derivative of m_n^{conv} and d_n^{conv} with respect to $\mathbf{W}_n^{(PQ)}$ can be calculated as (superscript ^{conv} is omitted

to simplify the description),

$$\frac{\partial w_n^{(PQ)}}{\partial m_n} = \begin{cases} 0, |w_n^{(P)}| \leq m_n \\ \mathcal{S}(w_n^{(P)}), |w_n^{(P)}| > m_n \end{cases} \tag{12}$$

$$\frac{\partial w_n^{(PQ)}}{\partial d_n} = \begin{cases} \frac{1}{d_n}(w_n^{(PQ)} - w_n^{(P)}), |w_n^{(P)}| \leq m_n \\ 0, |w_n^{(P)}| > m_n \end{cases}, \tag{13}$$

where $|\cdot|$ computes the absolute value and the scalar $w_n^{(P)}$ is one of parameters in the kernel $\mathbf{W}_n^{(P)}$.

Notice that, bit_n^{conv} is restricted to be an integer, d_n^{conv} and m_n^{conv} are restricted to be a value of 2^i, where i is an integer. We keep floating-point values during the back propagation and apply STE (straight through estimation) [17] to calculate the gradients. We hence could train the hypernetwork through end-to-end training and optimize quantization parameters Θ_n and Ω_{n-1}. In the forward propagation, they are rounded up to the closest required values, which are hence applied to quantize weights and activations with Eq. (2).

4.3 Iterative Compression Constraint for Training

The hypernetwork is trained from scratch. As a higher compression rate R commonly corresponds to lower network performance, directly applying the target compression rate R for training may lead to poor performance. To address this issue, we adopt an iterative compression constraint during training. During the searching stage, we gradually increase the target compression rate to allow the compressed network to maintain high performance. Also, following [7], we use log of the BOPs values as regularization term for network training. This is implemented by the $\mathcal{L}_{REG}$, *i.e.*,

$$\mathcal{L}_{REG} = \log(\max((\sum_{n=1:N} O_n)/(r \cdot \hat{O}), 1)), \tag{14}$$

where r is deceased linearly from 100% to the target value R by the end of searching stage. We apply Eq. (4) to estimate the BOPs of each layer. To approximately estimate the channel number $\hat{c_n}$ after pruning from α_n, we apply the Sigmoid function on α_n, *i.e.*,

$$\hat{c_n} = \sum_{i=1:c_n} \text{Sigmoid}(\alpha_n[i]). \tag{15}$$

5 EXPERIMENT

5.1 Datasets

To evaluate the effectiveness of our method, we conduct experiments on image classification and single image super-resolution tasks.

For classification tasks, we conduct experiments on both CIFAR-10 [18] and Tiny-ImageNet [5]. CIFAR-10 consists of 60k 32×32 sized images in 10 classes, including 50k training images and 10k test images. Tiny-ImageNet consists of 120k 64×64 sized images in 200 classes, including 100k training images, 10k validation images, and 10k test images.

As for image super-resolution task, DIV2K [1] is used as training set. It contains 800 training images, 100 validation images, and 100 test images, including 100k training images, 10k validation images, and 10k test images. Image patches are extracted from the training images. Following [22], the compressed networks are then tested on Set5 [3], Set14 [42], B100 [27], Urban100 [16], and DIV2K [1] validation set.

5.2 Implementation Details

For each convolutional neural network, we first initialize all latent vectors with standard normal distribution. For the hypernetworks, all biases are initialized as zero. Also, following [22], we initialize $\mathbf{H}_{n_1}$ and $\mathbf{H}_{n_2}$, $n = 1 : N$ with Xavier uniform and Hyperfan-in respectively. During searching, latent vectors and quantization parameters are updated to reach the constraint target. After the constraint target is met, the searching stage stops and quantization parameters are fixed. Then, only channels with latent vector values above the threshold are kept, and the rest are pruned. The network is then initialized with weights calculated by hypernetworks, and fine-tuned to get a better performance. For classification task, cross-entropy loss is applied. For super-resolution task, l_1 loss is applied following [22].

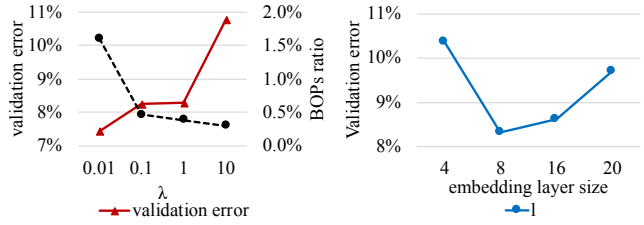
During training, we notice that compressing to 25% of its size using quantization, *i.e.*, from 32 bits to 8 bits bring marginal accuracy degradation. While compressing a network to 50% of its size using pruning may cause obvious accuracy reduction. We therefore initialize the quantization parameters as 8 bits to seek a more reasonable initialization. In this way, models can be pruned and quantized into reasonable structures.

VGG-small on CIFAR-10: We conduct experiments on CIFAR-10 using VGG-small [38]. We pad 4 pixels on each side of the input image and crop it randomly into the size of 32×32. Then, all images are scaled into the range [-1, 1]. We set batch size as 128, initial learning rate as 0.003 at searching stage and 0.01 at fine-tuning stage. During the searching stage, we set epochs as 100, R as 0.004. We reduce the learning rate by 0.1 at epoch 50. The fine-tuning stage is trained it for 300 epochs and we reduce the learning rate by 0.1 at epoch 150 and 225 respectively.

ResNet-20 on CIFAR-10: We conduct experiments on CIFAR-10 using ResNet-20. We set batch size as 128, initial learning rate as 0.001 at searching stage and 0.01 at fine-tuning stage. During the searching stage, we set epochs as 100, R as 0.009. We reduce the learning rate by 0.1 at epoch 50. The fine-tuning stage is trained for 300 epochs, and we reduce the learning rate by 0.1 at epoch 150 and 225 respectively.

MobileNetV2 on Tiny-ImageNet: We conduct experiments on Tiny-ImageNet using MobileNetV2. We set batch size as 128, initial learning rate as 0.01 at searching stage and 0.1 at fine-tuning stage. During the searching stage, we set epochs as 100, R as 0.05 and 0.1 for two settings. We reduce learning rate by 0.1 at epoch 30 and 60 respectively. The fine-tuning stage is trained for 220 epochs, and we reduce learning rate by 0.1 at epoch 200 and 210 respectively.

EDSR on Image super-resolution task: We conduct experiments on image super-resolution task base on EDSR [24]. We set batch size as 16, initial learning rate as 0.001 at searching stage and 0.01 at fine-tuning stage. During the searching stage, we set epochs as 100 and reduce the learning rate by 0.1 at epoch 50. During the



(a) Classification error and BOPs compression rates with different settings of λ.

(b) Classification error with different settings of l.

Figure 3: The experimnets are conducted on CIFAR-10 with VGG-small.

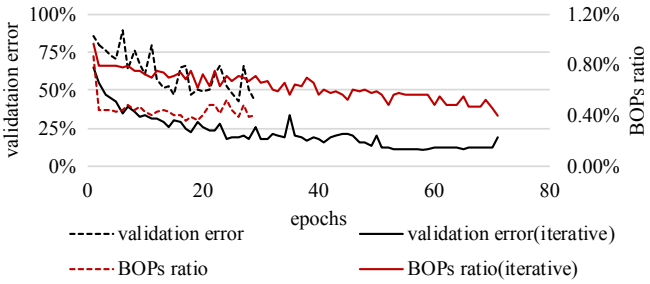


Figure 4: Illustration of training curves of our method with and without iterative compression constraint, respectively.

fine-tuning stage, we train it for 300 epochs and reduce the learning rate by 0.1 at epoch 40 and 200 respectively.

5.3 Ablation Study

This part first analyzes the setting of weight λ of $\mathcal{L}_{reg}$ and the setting of embedding layer size l. Then, we test the validity of the iterative compression constraint training strategy. Finally, we discuss the adaptive compression rates learned for different layers and neural networks.

The effect of regulation loss weight λ: Larger regulation loss weight can lead to faster convergence in the searching stage. However, simply meeting the constraint target without parameters training may lead to unreasonable structures. The impacts of different loss weights on the compressed network performance are summarized in Fig. 3 (a). We set the constraint target R as 0.004. As shown in the figure, setting loss weights too large may lead to unreasonable compression combinations. Setting a too small λ makes it hard to meet the BOPs constraint target. Too large λ is harmful to the accuracy. According to Fig. 3 (a), we fix λ as 1 in the following experiments. Notice that though this ablation study is conducted on CIFAR-10, further experiments in different neural network architectures and tasks using λ as 1 achieve high performance. Performances are shown in Sec. 5.4.

The effect of embedding layer size l: The size of the hypernetworks are decided by embedding layer size l. On the one hand, setting l too small could impair the function of hypernetworks. On the other hand, setting l too large would make the training process more difficult to converge while costing more training time. We therefore conduct ablation study on embedding layer size l and

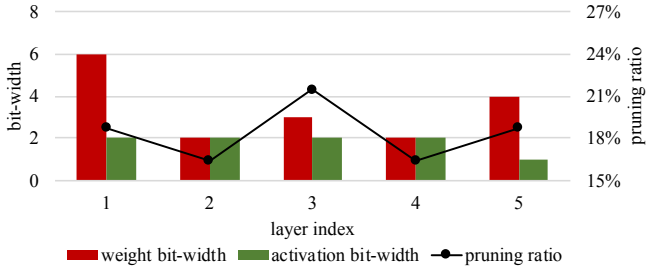


Figure 5: Illustration of learned compression parameters $\mathbb{A}$, $\mathbb{B}$, and $\mathbb{C}$ for VGG-small trained on CIFAR-10.

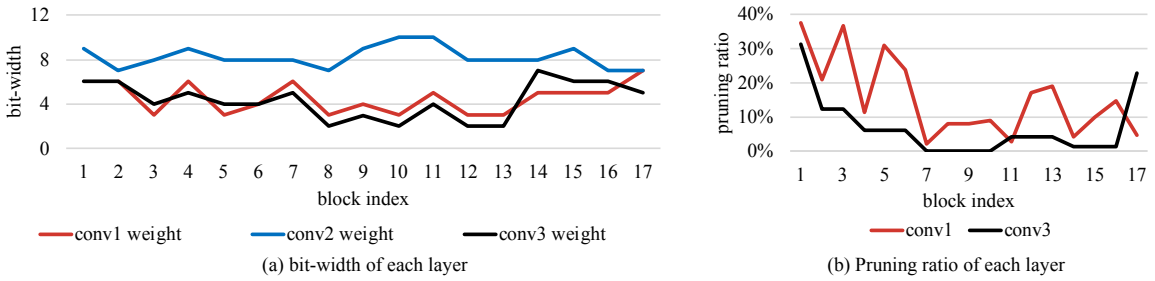
visualize the result in Fig. 3 (b). According to Fig. 3 (b), we fix l as 8 in the following experiments.

The effect of iterative compression constraint: Fig. 4 illustrates the training without iterative compression constraint. The network is quickly optimized to satisfy the target BOPs constraint. Meanwhile, it gets poor classification accuracy. Therefore, directly applying the target compression rate R leads to poor performance. With the iterative compression constraint, our method achieves the error rate of 13.12% in searching stage and error rate of 8.32% after fine-tuning. Meanwhile, it satisfies the 0.40% BOPs ratio constraint. It is clear that, the method without iterative compression constraint only reaches an error rate of 42.56%. And it is hard to converge in fine-tuning stage. The above comparisons thus have shown the validity of our iterative compression constraint.

Discussion on learned compression rates for different layers: Fig. 5 and Fig. 6 show learned compression parameters for two different networks. It is interesting to observe that, those two networks get different compression parameters. For VGG-small, our method (denoted as HNC) can get small quantization bit-with and large pruning ratio. This is because the VGG-small exists redundancy on the classification task. Notice that the first and the last convolutional layers preserve high bit-width for the weights. This indicates that the reducing bit-width of the first and the last layers would bring high performance loss. Similar results can also be drawn from works [17, 38, 43].

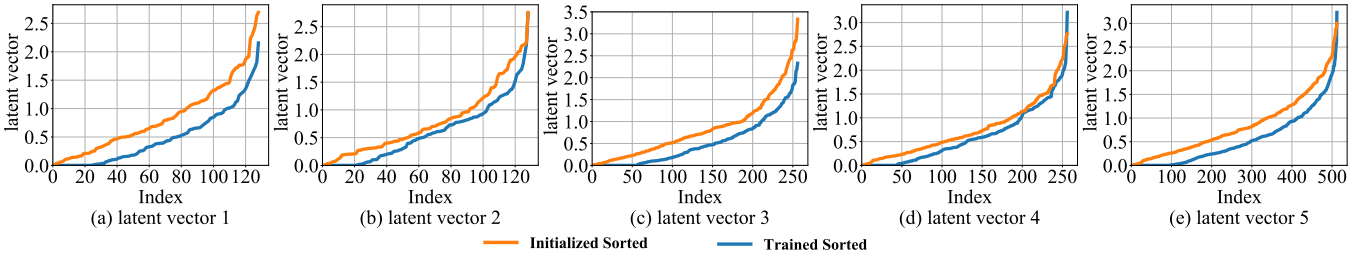
For MobileNetV2, HNC is able to compress many layers with small bit-width and a large pruning ratio. Notice that, MobileNetV2 [30] leverages linear bottlenecks to build the network. For each block, the feature map is first expanded in channel by a 1×1 convolutional layers. Then, a 3×3 channel-wise convolutional layer is applied to capture the spatial information of the feature map. Finally, a 1×1 convolutional layers is applied to get the output of a block. As shown in Fig. 6, each 3×3 channel-wise convolutional layer maintains high bit-width of activations and weights. This could help the channel-wise convolutional capture necessary spatial information. For both conv1 and conv3, the main purposes of using these 1×1 convolutional layers are to capture channel interaction. We can observe that after compressed by HNC, the bit-width of 1×1 convolutional layers weights and activations are also reduced, but most of the channels are remained. This is consistent with the purposes to learn channel interaction.

For each linear bottleneck module, the pruning ratio of conv1 is higher than that of conv3. This could be because the expansion



(a) bit-width of each layer (b) Pruning ratio of each layer

Figure 6: Illustration of learned compression parameters bit-width of weights and pruning ratios of each block for MobileNetV2 trained on Tiny-ImageNet. Bit-width of weights and activations share similar pattern. MobileNetV2 is built by linear bottleneck modules. Each module consists of three convolutional layers (conv1: a 1×1 convolutional layer, conv2: a 3×3 lightweight channel-wise convolutional layer, conv3: a 1×1 convolutional layer). conv2 is a channel-wise convolutional layer which can not be pruned.



(a) latent vector 1 (b) latent vector 2 (c) latent vector 3 (d) latent vector 4 (e) latent vector 5

Figure 7: Illustration of initialized latent vectors and trained latent vectors of each convolutional layer in VGG-small.

Method	year	Bit-width	Test Error	BOPs(G)	MACs(G)
VGG-small					
Baseline	-	32/32	7.00%	623.12	0.613
TWN [20]	2016	2/32	7.44%	38.88	0.613
RQ [26]	2018	8/8	6.70%	38.88	0.613
RQ [26]	2018	4/4	7.96%	9.72	0.613
WAGE [39]	2018	2/8	6.78%	9.72	0.613
DQ [35]	2019	mixed	8.41%	3.03	0.613
DJPQ [38]	2020	mixed	8.46%	2.99	0.367
HNC	-	mixed	8.32%	**2.43**	0.487
ResNet-20					
Baseline	-	32/32	7.46%	42.20	0.042
Rethinking [41]	2018	32/32	9.10%	22.20	0.022
DHP-50 [22]	2020	32/32	8.46%	21.86	0.022
FPGM [14]	2019	32/32	9.38%	19.41	0.019
DQ [35]	2019	mixed	8.58%	-	0.042
HNC	-	mixed	7.99%	**3.45**	0.026

Table 1: Comparison on CIFAR-10 with VGG-small and ResNet-20. BOPs and MACs are reported to measure the compression ratios of the networks as in [38].

Method	year	Bit-width	Test Error	BOPs(G)	MACs(G)
baseline	-	32/32	44.75%	353.68	5.55
DHP-10 [22]	2020	32/32	52.43%	42.15	0.662
MetaPruning [25]	2019	32/32	56.72%	38.90	0.610
MobileNetV2-0.3 [30]	2018	32/32	53.99%	35.69	0.560
HNC	-	mixed	57.30%	**15.39**	4.12
HNC	-	mixed	**50.68%**	31.03	4.20

Table 2: Comparison on Tiny-ImageNet with MobileNetV2.

Method	year	Bit-width	Test Error	BOPs(G)	MACs(G)
Baseline	-	32/32	30.26%	1853.44	1.81
RQ [26]	2018	8/8	30.03%	115.84	1.81
TWN [20]	2016	2/32	28.20%	115.84	1.81
RQ [26]	2018	4/4	28.48%	28.96	1.81
DQ	2019	mixed	31.52%	40.71	1.81
DJPQ	2020	mixed	30.63%	35.01	1.39
DJPQ	2020	mixed	31.20%	30.87	1.39
HNC	-	mixed	**29.97%**	**30.75**	1.50

Table 3: Comparison on ImageNet with ResNet-18. The BOPs of DQ is not reported in the paper.

factor in conv1 exists redundancies. While the channel number of conv3 is already small. The compression results share similarity with that in DJPQ [38]. It tends to keep the bit-width of conv2 layers high. It also tends to quantize the conv1 and conv3 layers into relatively lower bit-width.

Compare with DJPQ [38], our method leverages meta-learning to learn the weights of each convolutional layer. In this way, our method can better apply pruning and quantization for different layers. Though experiments, we can observe that our method tends to apply pruning for all layers, while DJPQ scheme mainly focuses on the bottom layers.

Method	Set5	Set14	B100	Urban100	DIV2K	BOPs(T)
Baseline	32.1	28.55	27.55	26.02	28.93	92.5
clustering [32]	31.93	28.47	27.48	25.77	28.80	92.5
Factor-SIC3 [37]	31.96	28.47	27.49	25.81	28.81	67.1
Basic-128-40 [21]	32.03	28.45	27.50	25.81	28.82	64.2
Fastor-SIC2 [37]	31.82	28.40	27.43	25.63	28.70	62.4
Basic-128-27 [21]	31.95	28.42	27.46	25.76	28.76	59.7
DHP-60 [22]	31.99	28.52	27.53	25.92	28.88	57.0
DHP-40 [22]	32.01	28.49	27.52	25.86	28.85	38.7
DHP-20 [22]	31.94	28.42	27.47	25.69	28.77	19.9
HNC	30.58	27.62	26.94	24.53	28.61	**10.7**
HNC	31.98	28.43	27.49	25.72	28.79	17.6

Table 4: Comparison on Super-resolution task. Peak Signal-to-Noise Ratio (dB) is used to measure the practical effectiveness. The BOPs (TFLOPs) is also reported.

5.4 Comparison on Recent Works

Comparison on CIFAR-10 with VGG-small: We first compare our method (denoted as HNC) with recent works on CIFAR-10. The results are shown in Table 1. Fig. 5 also illustrates the learned pruning ratio, weight and activation bit-width by HNC. We compare HNC with both fixed-point quantization methods, like TWN [20], RQ [26] and WAGE [39] and mixed-point quantization methods, like DQ [35]. Also, we compare HNC with DJPQ [38] which combines pruning and quantization methods. Compared to the baseline model, HNC reduces the BOPs from 623.12G to 2.43G with only a 1.32% accuracy drop. Compared with those quantization methods, HNC achieves substantially better performance in aspects of both compression rate and accuracy. It is also clear that, compared with the recent DJPQ [38], our method achieves a higher compression rate and better accuracy. What's more, we visualize the initialized latent vectors and trained latent vectors in Fig. 7. As we can see, our method manages to reduce the values of unimportant channels value down to pretty low values and successfully prune these channels.

Comparison on CIFAR-10 with ResNet-20: To further validate the effectiveness of HNC, we conduct experiments on a different network structure ResNet-20. We compare HNC with other pruning methods, Rethinking [41], FPGM [14] and DHP-50 [22]. As shown in Table 1, compared with the baseline model, HNC reduces the BOPs from 42.2G to 3.45G with only 0.53% accuracy drop. The results show a similar conclusion with Table 1. Compared with those methods, HNC gets a lower pruning ratio, but gets better BOPs and accuracy. This indicates that, quantization could be a better choice for compressing ResNet-20. Table 1 also compares HNC with DQ [35], which combines pruning and quantization. HNC also achieves lower classification error than DQ.

Comparison on Tiny-ImageNet with MobileNetV2: For experiments on the Tiny-ImageNet, we applied our method to MobileNetV2 structure, which is light-weighted and designed for fast speed and high accuracy. As shown in Table 2, we compare HNC with [22], [25] and MobileNetV2-0.3 (compressed MobileNetV2 networks with 0.3 channel numbers). Compared with those compression methods, HNC achieves a significantly larger BOPs reduction. Especially when compared with MetaPruning, HNC achieves about 2.5× reduction in BOP counts (15.39G vs. 38.90G) with only 1.42% accuracy drop. Through the comparison, we show that HNC can automatically combine pruning and quantization to achieve a larger compression ratio. Also, by constraining the BOPs to 31.03G, HNC is able to outperform DHP-10 with 1.75% better efficiency.

Comparison on ImageNet with ResNet-18: For experiments on the ImageNet, we applied our method to ResNet-18 structure. As shown in Table 3, We achieve 70.03% accuracy with 30.75 GBOPs. While DJPQ gets 69.27% accuracy with 35.01 GBOPs, and 68.80% with 30.87 GBOPs. Our method achieves better accuracy with a higher compression rate.

Comparison on Super-resolution Task: To further validate the effectiveness of HNC, we apply HNC on super-resolution task and summarize the results in Table. 4. We compare HNC with clustering [32], Factor-SIC3, Fastor-SIC2 [37], Basic-128-40, Basic-128-27 [21] and DHP [22]. As we can see, HNC achieves comparable performance with other methods, meanwhile demonstrate substantial advantages on efficiency. With only (10.7/92.5)11.54% BOPs computation resources, HNC is able to achieve a high PSNR, only 1.52%dB lower than baseline. HNC also achieves 31.98dB with (17.6/92.5)19.03% BOPs Ratio on Set5. Compare with DHP-20 [22], HNC achieves better performance with a higher compression rate.

Discussions: Quantized CNN can obtain low theoretical BOPs by leveraging low bit-wise operations. However, the inference speed of networks also greatly depends on specific kernel-implementations. Therefore, it is unfair to directly compare the inference time on traditional GPUs. HNC applies structure pruning and quantization for the network, which have been proved are two of the most successful techniques in compressing a CNN for efficient inference [10, 13, 15]. Also, low bit-width operations can save energy for specific hardware according to [39]. Following several papers such as [17, 26, 38], we apply BOPs as the regulation target. Nevertheless, when it comes to different constrain targets, it would be easy to achieve different constraints by modifying the regularization loss.

6 CONCLUSION

This paper unifies quantization and pruning for CNN compression by generating set of layer-wise compression parameters with meta-learning. We use the proposed regularization loss to unify the channel numbers, precision of network activations and weights. An iterative compression constraint is also proposed to help the compression training of CNNs. We analyze the compression details of different network architectures and show the validity of our proposed method. We conduct many experiments with different network structures on different tasks. Extensive experiments demonstrate the superior performance of the proposed method in aspects of both accuracy and efficiency. We hence could conclude that, considering multiple criteria leads to better compression.

Acknowledgments This work is supported in part by Peng Cheng Laboratory, in part by The National Key Research and Development Program of China under Grant No. 2018YFE0118400, in part by Natural Science Foundation of China under Grant No. U20B2052, 61936011, 61620106009, in part by Beijing Natural Science Foundation under Grant No. JQ18012.

REFERENCES

[1] Eirikur Agustsson and Radu Timofte. 2017. Ntire 2017 challenge on single image super-resolution: Dataset and study. In *CVPRW*.

[2] Chaim Baskin, Eli Schwartz, Evgenii Zheltonozhskii, Natan Liss, Raja Giryes, Alexander M. Bronstein, and Avi Mendelson. 2018. UNIQ: Uniform Noise Injection for the Quantization of Neural Networks. *CoRR* (2018).

[3] Marco Bevilacqua, Aline Roumy, Christine Guillemot, and Marie Line Alberi-Morel. 2012. Low-complexity single-image super-resolution based on nonnegative neighbor embedding. (2012).

[4] Bin Dai, Chen Zhu, Baining Guo, and David Wipf. 2018. Compressing neural networks using the variational information bottleneck. In *ICML*.

[5] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *CVPR*.

[6] Ahmed T Elthakeb, Prannoy Pilligundla, FatemehSadat Mireshghallah, Amir Yazdanbakhsh, Sicun Gao, and Hadi Esmaeilzadeh. 2018. Releq: an automatic reinforcement learning approach for deep quantization of neural networks. *arXiv* (2018).

[7] Shaopeng Guo, Yujie Wang, Quanquan Li, and Junjie Yan. 2020. DMCP: Differentiable Markov Channel Pruning for Neural Networks. In *CVPR*.

[8] Philipp Gysel, Jon Pimentel, Mohammad Motamedi, and Soheil Ghiasi. 2018. Ristretto: A framework for empirical study of resource-efficient inference in convolutional neural networks. *TNNLS* (2018).

[9] Song Han, Huizi Mao, and William J Dally. 2016. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *ICLR* (2016).

[10] Song Han, Huizi Mao, and William J Dally. 2016. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *ICLR* (2016).

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *CVPR*.

[12] Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. 2018. Soft filter pruning for accelerating deep convolutional neural networks. In *IJCAI*.

[13] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. 2018. AMC: Automl for model compression and acceleration on mobile devices. In *ECCV*.

[14] Yang He, Ping Liu, Ziwei Wang, Zhilan Hu, and Yi Yang. 2019. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *CVPR*.

[15] Yihui He, Xiangyu Zhang, and Jian Sun. 2017. Channel pruning for accelerating very deep neural networks. In *ICCV*.

[16] Jia-Bin Huang, Abhishek Singh, and Narendra Ahuja. 2015. Single image super-resolution from transformed self-exemplars. In *CVPR*.

[17] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized Neural Networks. In *NeurIPS*.

[18] Alex Krizhevsky. 2009. Learning Multiple Layers of Features from Tiny Images. (2009).

[19] Se Jung Kwon, Dongsoo Lee, Byeongwook Kim, Parichay Kapoor, Baeseong Park, and Gu-Yeon Wei. 2020. Structured Compression by Weight Encryption for Unstructured Pruning and Quantization. In *CVPR*.

[20] Fengfu Li, Bo Zhang, and Bin Liu. 2016. Ternary weight networks. *arXiv* (2016).

[21] Yawei Li, Shuhang Gu, Luc Van Gool, and Radu Timofte. 2019. Learning filter basis for convolutional neural network compression. In *ICCV*.

[22] Yawei Li, Shuhang Gu, Kai Zhang, Luc Van Gool, and Radu Timofte. 2020. DHP: Differentiable Meta Pruning via HyperNetworks. In *ECCV*.

[23] Yawei Li, Wen Li, Martin Danelljan, Kai Zhang, Shuhang Gu, Luc Van Gool, and Radu Timofte. 2021. The Heterogeneity Hypothesis: Finding Layer-Wise Differentiated Network Architectures. In *CVPR*.

[24] Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, and Kyoung Mu Lee. 2017. Enhanced deep residual networks for single image super-resolution. In *CVPRW*.

[25] Zechun Liu, Haoyuan Mu, Xiangyu Zhang, Zichao Guo, Xin Yang, Kwang-Ting Cheng, and Jian Sun. 2019. Metapruning: Meta learning for automatic neural network channel pruning. In *ICCV*.

[26] Christos Louizos, Matthias Reisser, Tijmen Blankevoort, Efstratios Gavves, and Max Welling. 2018. Relaxed Quantization for Discretized Neural Networks. In *ICLR*.

[27] David Martin, Charless Fowlkes, Doron Tal, and Jitendra Malik. 2001. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *ICCV*. IEEE.

[28] 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* (2016).

[29] Hanyu Peng, Jiaxiang Wu, Shifeng Chen, and Junzhou Huang. 2019. Collaborative channel pruning for deep networks. In *ICML*.

[30] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*.

[31] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. In *ICLR*.

[32] Sanghyun Son, Seungjun Nah, and Kyoung Mu Lee. 2018. Clustering convolutional kernels to compress deep neural networks. In *ECCV*.

[33] Lucas Theis, Iryna Korshunova, Alykhan Tejani, and Ferenc Huszár. 2018. Faster gaze prediction with dense networks and fisher pruning. *arXiv preprint arXiv:1801.05787* (2018).

[34] Naftali Tishby, Fernando C Pereira, and William Bialek. 1999. The information bottleneck method. In *Annual Allerton Conference on Communications, Control and Computing*.

[35] Stefan Uhlich, Lukas Mauch, Fabien Cardinaux, Kazuki Yoshiyama, Javier Alonso Garcia, Stephen Tiedemann, Thomas Kemp, and Akira Nakamura. 2019. Mixed Precision DNNs: All you need is a good parametrization. In *ICLR*.

[36] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. 2019. Haq: Hardware-aware automated quantization with mixed precision. In *CVPR*.

[37] Min Wang, Baoyuan Liu, and Hassan Foroosh. 2017. Factorized convolutional neural networks. In *ICCVW*.

[38] Ying Wang, Yadong Lu, and Tijmen Blankevoort. 2020. Differentiable joint pruning and quantization for hardware efficiency. In *ECCV*.

[39] Shuang Wu, Guoqi Li, Feng Chen, and Luping Shi. 2018. Training and inference with integers in deep neural networks. *arXiv* (2018).

[40] Haichuan Yang, Shupeng Gui, Yuhao Zhu, and Ji Liu. 2020. Automatic Neural Network Compression by Sparsity-Quantization Joint Learning: A Constrained Optimization-Based Approach. In *CVPR*.

[41] Jianbo Ye, Xin Lu, Zhe Lin, and James Z Wang. 2018. Rethinking the Smaller-Norm-Less-Informative Assumption in Channel Pruning of Convolution Layers. In *ICLR*.

[42] Roman Zeyde, Michael Elad, and Matan Protter. 2010. On single image scale-up using sparse-representations. In *ICCS*. Springer.

[43] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. 2016. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv* (2016).