LBS: Loss-aware Bit Sharing for Automatic Model Compression

Jing Liu [1] **Bohan Zhuang** [1] **Peng Chen** [1] **Yong Guo** [2] **Chunhua Shen** [1] **Jianfei Cai** [1] **Mingkui Tan** [2]

Abstract

Low-bitwidth model compression is an effective method to reduce the model size and computational overhead. Existing compression methods rely on some compression configurations (such as pruning rates, and/or bitwidths), which are often determined manually and not optimal. Some attempts have been made to search them automatically, but the optimization process is often very expensive. To alleviate this, we devise a simple yet effective method named Loss-aware Bit Sharing (LBS) to automatically search for optimal model compression configurations. To this end, we propose a novel single-path model to encode all candidate compression configurations, where a high bitwidth quantized value can be decomposed into the sum of the lowest bitwidth quantized value and a series of re-assignment offsets. We then introduce learnable binary gates to encode the choice of bitwidth, including filter-wise 0-bit for filter pruning. By jointly training the binary gates in conjunction with network parameters, the compression configurations of each layer can be automatically determined. Extensive experiments on both CIFAR-100 and ImageNet show that LBS is able to significantly reduce computational cost while preserving promising performance.

1. Introduction

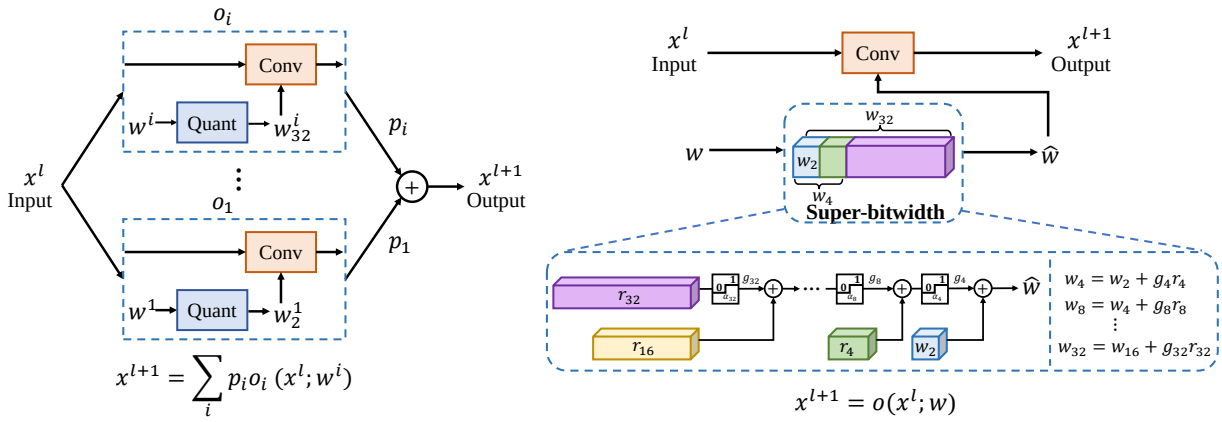
Deep neural networks (DNNs) (LeCun et al., 1989) have achieved great success in many challenging computer vision tasks, including image classification (Krizhevsky et al., 2012; He et al., 2016) and object detection (Lin et al., 2017a;b). However, a deep model usually has a large number of parameters and consumes enormous computational resources, which presents great obstacles for many applications, especially on resource-limited devices with limited computational resources, such as smartphones. To reduce the number of parameters and computational overhead, many methods (He et al., 2019; Zhou et al., 2016) have been proposed to conduct model compression by removing the redundancy while maintaining the performance.

In recent years, we have witnessed the significant progress of model compression methods. Specifically, network quantization (Zhou et al., 2016; Hubara et al., 2016) reduces the model size and computational cost by mapping the full-precision values to low-precision ones. Conventional quantization methods (Zhou et al., 2016; Choi et al., 2018) use the same bitwidth for all layers of a network. Nevertheless, different layers have different redundancy and contribute differently to the accuracy and efficiency of a network. To obtain better performance, it is necessary to consider mixed-precision quantization that assigns different bitwidths to different layers, which has been supported by many hardware platforms, such as NVIDIA Turing GPU (Burgess, 2019) and FPGAs (Sharma et al., 2018).

Given a deep neural network, the search space grows exponentially with the number of compression configurations. To efficiently explore the space, many attempts have been proposed, either based on reinforcement learning (RL) (Wang et al., 2019), evolutionary search (ES) (Wang et al., 2020), Bayesian optimization (BO) (Tung & Mori, 2018) or differentiable methods (Wu et al., 2018). In particular, previous differentiable methods formulate the mixed-precision quantization problem as a path selection problem and explore the exponential search space using gradient-descent optimization. As shown in Figure 1(a), these methods construct a multi-path heavy network wherein each candidate compression operation is maintained as a separate path. However, these methods suffer from a critical issue. When the search space becomes combinatorially large, the multi-path scheme leads to a huge amount of parameters and high computational cost. Due to the parameter explosion introduced by the multi-path scheme, the optimization of the heavily compressed candidate network would be challenging.

In this paper, we propose a simple yet effective method named Loss-aware Bit Sharing (LBS) to reduce the search cost and ease the optimization of the compressed candidates. Motivated by recent neural architecture search (NAS) methods (Stamoulis et al., 2019; Guo et al., 2020), LBS introduces a novel single-path bit sharing model to encode

[1]Monash University [2]South China University of Technology. Correspondence to: Bohan Zhuang <bohan.zhuang@monash.edu>.



(a) Multi-path search scheme (Wu et al., 2018)

(b) Single-path search scheme (Ours)

Figure 1. Multi-path vs. single-path compression scheme. (a) Multi-path search scheme (Wu et al., 2018): represents each candidate configuration as a separate path and formulates the mixed-precision quantization problem as a path selection problem, which gives rise to huge numbers of trainable parameters and high computational overhead when the search space becomes large. Here, o_i and p_i are the convolutional operation and probability for the i-th configuration, respectively. x^l indicates the input feature maps for layer l, w^i is the weight for the i-th configuration and w_k^i is the k-bit quantized version of w^i. (b) Single-path search scheme (Ours): represents each candidate configuration as a subset of a "super-bitwidth" and formulates the mixed-precision quantization problem as a subset selection problem, which greatly reduces the number of parameters, computational cost, and optimization difficulty. Here, the super-bitwidth denotes the highest bitwidth (*i.e.*, 32-bit) in the search space, w is the weight for the convolutional operation, w_k is the k-bit quantized version of w, r_k is the re-assignment offset and g_k is the corresponding binary gate to controls the bitwidth decision.

all bitwidths in the search space, as shown in Figure 1(b). Our proposed method is built upon the observation that the quantized values of a high bitwidth can be shared with those of low bitwidths under some conditions. Therefore, we are able to decompose the quantized representation into the sum of the lowest bitwidth representation and a series of re-assignment offsets. In this way, the quantized values of different bitwidths can be viewed as different subsets of those of super-bitwidth (*i.e.*, the highest bitwidth in the search space). Instead of choosing among different paths in the multi-path scheme, the proposed LBS solves the mixed-precision quantization problem by finding the appropriate subset of quantized values to use, which greatly reduces the number of parameters, computational cost, and optimization difficulty of the compressed candidates. We then introduce learnable binary gates to encode the choice of bitwidth, including filter-wise 0-bit for pruning. By jointly training the binary gates and network parameters, the bitwidth of each layer can be automatically determined.

Our main contributions are summarized as follows:

- We devise a novel single-path scheme that encapsulates multiple compression configurations in a unified single-path framework. Relying on this, we further introduce learnable binary gates to determine the optimal bitwidths (including filter-wise 0-bit for pruning). The proposed LBS casts the search problem as a subset selection problem, hence significantly reducing the number of parameters and search cost.
- We formulate the quantized representation as a gated combination of the lowest bitwidth representation and a series of re-assignment offsets, in which we explicitly share the quantized values between different bitwidths. In this way, we enable the candidate operations to learn jointly rather than separately, hence greatly easing the optimization of the compressed candidate network.
- We evaluate our LBS on CIFAR-100 and ImageNet over various network architectures. Extensive experiments show that the proposed method achieves the state-of-the-art performance. For example, on ImageNet, our LBS compressed MobileNetV2 achieves 22.6× Bit-Operation (BOP) reduction with only 0.1% performance decline in terms of the Top-1 accuracy.

2. Related Work

Network quantization. Network quantization represents the weights, activations and even gradients with low precision to yield compact DNNs. With low-precision integers or power-of-two representations, the heavy matrix multiplications can be replaced by efficient bitwise operations, leading to much faster test-time inference and lower power consumption. To improve the quantization performance, existing methods either focus on learning accurate quantizers (Jung et al., 2019; Zhang et al., 2018; Choi et al., 2018; Cai et al., 2017), or seek to approximate the gradients due to the non-differentiable discretization (Ding et al., 2019;

Louizos et al., 2019; Zhuang et al., 2020). Moreover, most previous works assign the same bitwidth for all layers (Zhou et al., 2016; Zhuang et al., 2018a; 2019; Jung et al., 2019; Jin et al., 2019; Li et al., 2020; Esser et al., 2020). Though attractive for simplicity, setting a uniform precision places no guarantee on optimizing network performance, since different layers have different redundancy and arithmetic intensity. Therefore, several studies proposed mixed-precision quantization (Wang et al., 2019; Dong et al., 2019; Wu et al., 2018; Uhlich et al., 2020) to set different bitwidths according to the redundancy of each layer. In this paper, based on the proposed single-path bit sharing decomposition, we devise an approach that can effectively learn appropriate bitwidths for different layers through gradient-based optimization.

NAS and pruning. Neural architecture search (NAS) aims to automatically design efficient architectures, either based on reinforcement learning (Pham et al., 2018; Guo et al., 2019), evolutionary search (Real et al., 2019) or gradient-based methods (Liu et al., 2019a; Cai et al., 2019a). In particular, gradient-based NAS has gained increased popularity, where the search space can be divided into the multi-path design (Liu et al., 2019a; Cai et al., 2019b) and single-path formulation (Stamoulis et al., 2019; Guo et al., 2020), depending on whether each operation is added as a separate path or not. While prevailing NAS methods optimize the network topology, the focus of this paper is to search optimal compression ratios for a given architecture. Moreover, network pruning can be treated as fine-grained NAS, which aims at removing redundant modules to reduce the model size and accelerate the run-time inference speed, giving rise to methods based on unstructured weight pruning (Han et al., 2016; Guo et al., 2016) or structured channel pruning (He et al., 2017; Zhuang et al., 2018b; Luo et al., 2017). Based on channel pruning, our paper further takes quantization into consideration to generate more compact networks.

AutoML for model compression. Recently, much effort has been devoted to automatically determining either the pruning rate (Tung & Mori, 2018; Dong & Yang, 2019; He et al., 2018), or the bitwidth (Lou et al., 2019; Cai & Vasconcelos, 2020) of each layer. In particular, HAQ (Wang et al., 2019) employs reinforcement learning to search optimal bitwidth with the hardware accelerator's feedback. Meta-pruning (Liu et al., 2019b) uses meta-learning to generate the weight parameters of the pruned networks and then adopts evolutionary search to find the layer-wise sparsity for channel pruning. More recently, several studies (Wu et al., 2018; Cai & Vasconcelos, 2020) have focused on using differentiable schemes via gradient-based optimization.

Closely related methods. Recently, several methods have been proposed to jointly optimize pruning and quantization strategies. In particular, some works only support weight quantization (Tung & Mori, 2018; Ye et al., 2019) or use fine-grained pruning (Yang et al., 2020). However, the resultant networks cannot be implemented efficiently on edge devices. To handle this, several methods (Wu et al., 2018; Wang et al., 2020; Ying et al., 2020) have been proposed to consider filter pruning, weight quantization, and activation quantization jointly. In contrast with these methods, we carefully design the compression search space by sharing the quantized values between different candidate configurations, which significantly reduces the search cost and eases the optimization. Compared with those methods that share the similarities of using quantized residual errors (Chen et al., 2010; Gong et al., 2014; Li et al., 2017b; van Baalen et al., 2020), our proposed method recursively uses quantized residual errors to decompose a quantized representation into a set of candidate bitwidths and parameterize the selection of optimal bitwidth via binary gates.

Compared with Bayesian Bits (van Baalen et al., 2020), our proposed LBS is developed concurrently that share a similar idea of quantization decomposition. Critically, our LBS differs from Bayesian Bits in several aspects: 1) The quantization decomposition in our method can be extended to non-power-of-two bitwidths (i.e., b_1 and γ_j can be set to arbitrary appropriate integer values), which is a general case of the one in Bayesian Bits. 2) The optimization problems are different. Specifically, we formulate model compression as a single-path subset selection problem while Bayesian Bits casts the optimization of the binary gates to a variational inference problem that requires more relaxations and hyperparameters. 3) Our compressed models with less or comparable BOPs outperform those of Bayesian Bits by a large margin on ImageNet (See Table 2).

3. Preliminary

For convenience, we revisit the normalization and quantization function. Without loss of generality, given a convolutional layer, let x and w be the activations of the last layer and its weight parameters, respectively. First, for convenience, following (Choi et al., 2018; Bai et al., 2019), we can normalize x and w into scale [0, 1] by T_x and T_w, respectively:

$$z_x = T_x(x) = \text{clip}\left(\frac{x}{v_x}, 0, 1\right), \tag{1}$$

$$z_w = T_w(w) = \frac{1}{2}\left(\text{clip}\left(\frac{w}{v_w}, -1, 1\right) + 1\right), \tag{2}$$

where the function $\text{clip}\left(v, v_{\text{low}}, v_{\text{up}}\right) = \min(\max(v, v_{\text{low}}), v_{\text{up}})$ clips any number v into the range $[v_{\text{low}}, v_{\text{up}}]$, and v_x and v_w are trainable quantization intervals which indicate the range of weights and activations to be quantized. Then, we can apply the following function to quantize the normalized activations and parameters,

namely $z_x \in [0, 1]$ and $z_w \in [0, 1]$, to discretized ones:

$$D(z, s) = s \cdot \text{round}\left(\frac{z}{s}\right), \tag{3}$$

where s denotes the normalized step size, $\text{round}(x) = \lceil x - 0.5 \rceil$ returns the nearest integer of a given value x and $\lceil \cdot \rceil$ is the ceiling function. Typically, for k-bit quantization, the normalized step size s can be computed by

$$s = \frac{1}{2^k - 1}. \tag{4}$$

After performing the k-bit quantization, we shall have $2^k - 1$ quantized values (except zero). Specifically, we obtain the quantizations of $Q(w)$ and $Q(x)$ by

$$Q(w) = T_w^{-1}(D(z_w, s)) = v_w \cdot (2 \cdot D(z_w, s) - 1), \tag{5}$$

$$Q(x) = T_x^{-1}(D(z_x, s)) = v_x \cdot D(z_x, s), \tag{6}$$

where T_w^{-1} and T_x^{-1} denote the inverse functions of T_w and T_x, respectively.

In this paper, we focus on the mixed-precision quantization problem that aims to find optimal bitwidths for different layers. To solve this, one can consider different bitwidths as different paths and reformulate the mixed-precision problem as a path selection problem (Wu et al., 2018), as shown in Figure 1(a). However, when the search space becomes large, it suffers from a huge number of parameters and high computational cost. As a result, the optimization of the compressed model becomes more challenging.

4. Proposed Method

In this paper, we propose a simple yet effective method to reduce the search cost and ease the optimization for the compressed candidates. In the following, we first devise a novel single-path bit sharing model to encode all bitwidths in the search space, where a high bitwidth quantized representation can be decomposed into the sum of the lowest bitwidth representation and a series of recursive re-assignment offsets. Then, we introduce learnable binary gates to encode the choice of bitwidth, including filter-wise 0-bit for pruning. Last, we propose to jointly train the binary gates and model parameters for loss-aware compression. For convenience, we call our method Loss-aware Bit Sharing (LBS).

4.1. Single-path Bit Sharing Decomposition

To illustrate the single-path bit sharing decomposition, we begin with an example of 2-bit quantization for $z \in \{z_x, z_w\}$. Specifically, we use the following equation to quantize z to 2-bit:

$$z_2 = D(z, s_2), \quad s_2 = \frac{1}{2^2 - 1}, \tag{7}$$

where z_2 and s_2 are the quantized value and the step size of 2-bit quantization, respectively. Due to the large step size, the residual error $z - z_2 \in [-s_2/2, s_2/2]$ may be large and results in a significant performance decline. One intuitive way to reduce the residual error is to use a smaller step size, which indicates that we quantize z to a higher bitwidth. Since the step size $s_4 = 1/(2^4 - 1)$ in 4-bit quantization is a divisor of the step size s_2 in 2-bit quantization, the quantized values of 2-bit quantization are shared with those of 4-bit quantization. In fact, based on 2-bit quantization, the 4-bit counterpart introduces additional unshared quantized values. In particular, if z_2 has zero residual error, then 4-bit quantization maps z to the shared quantized values (*i.e.*, z_2). In contrast, if z_2 has non-zero residual error, 4-bit quantization is likely to map z to the unshared quantized values. In this case, 4-bit quantization can be regarded as performing quantized value re-assignment based on z_2. Such a re-assignment process can be formulated as follows:

$$z_4 = z_2 + r_4, \tag{8}$$

where z_4 is the 4-bit quantized value and r_4 is the re-assignment offset based on z_2. To ensure that the results of re-assignment fall into the unshared quantized values, the re-assignment offset r_4 must be an integer multiple of the 4-bit step size s_4. Formally, r_4 can be computed by performing 4-bit quantization on the residual error of z_2:

$$r_4 = D(z - z_2, s_4), \quad s_4 = \frac{1}{2^4 - 1}. \tag{9}$$

According to Eq. (8), a 4-bit quantized value can be decomposed into the 2-bit representation and its re-assignment offset. Similarly, an 8-bit quantized value can also be decomposed into the 4-bit representation and its corresponding re-assignment offset. In this way, we can generalize the idea of decomposition to arbitrary effective bitwidths as follows.

Theorem 1 (Quantization Decomposition) *Let $z \in [0, 1]$ be a normalized full-precision input, and $\{b_j\}_{j=1}^K$ be a sequence of candidate bitwidths. If the bitwidth b_j is an integer multiple of b_{j-1},* i.e., *$b_j = \gamma_j b_{j-1} (j > 1)$, where $\gamma_j \in \mathbb{Z}^+ \backslash \{1\}$ is a multiplier, then the following quantized approximation z_{b_K} can be decomposed as:*

$$\begin{aligned} z_{b_K} &= z_{b_1} + \sum_{j=2}^{K} r_{b_j}, \\ \text{where } r_{b_j} &= D(z - z_{b_{j-1}}, s_{b_j}), \\ z_{b_j} &= D(z, s_{b_j}), \\ s_{b_j} &= \frac{1}{2^{b_j} - 1}. \end{aligned} \tag{10}$$

Theorem 1 indicates that the quantized approximation z_K can be decomposed into the sum of the lowest bitwidth

representation z_{b_1} and a series of recursive re-assignment offsets. To measure the quality of the quantization, we introduce the following corollary.

Corollary 1 (Normalized Quantization Error Bound) *Given $\mathbf{z} \in [0,1]^d$ being a normalized full-precision vector, $\mathbf{z}_{b_K}$ being its quantized vector with bitwidth b_K, let $\epsilon_K = \frac{\|\mathbf{z}-\mathbf{z}_{b_K}\|_1}{\|\mathbf{z}\|_1}$ be the normalized quantization error, then the following error bound w.r.t. K holds:*

$$|\epsilon_K - \epsilon_{K+1}| \leq \frac{C}{2^{b_K}-1}, \tag{11}$$

where $C = \frac{d}{\|\mathbf{z}\|_1}$ is a constant.

Corollary 1 implies that the normalized quantization error bound decreases quickly as the bitwidth increases. In other words, the error changes more slowly with higher bitwidth. The proof and more details can be found in the supplementary material.

Let the super-bitwidth (*i.e.*, 32-bit) be the highest bitwidth in the search space. From Theorem 1, the quantized values of different bitwidths in the search space can be viewed as different subsets of those of super-bitwidth. In this way, we only need to maintain a single-path model that encapsulates all the bitwidths in the search space, which greatly reduces the number of trainable parameters and search cost. Moreover, we are able to extract a low-precision representation from its higher precision, which allows for optimizing different bitwidths jointly and easing the optimization.

4.2. Single-path Bit Sharing Compression

In a neural network, different layers have different redundancy and contribute differently to the accuracy and efficiency. To determine the bitwidth for each layer, we introduce a layer-wise binary quantization gate $g_{b_j} \in \{0,1\} (j > 1)$ on each of the re-assignment offsets in Eq. (10) to encode the choice of the quantization bitwidth as:

$$\begin{aligned} g_{b_j} &= \mathbb{1}\left(||z - z_{b_{j-1}}|| - \alpha_{b_j} > 0\right), \\ z_{b_K} &= z_{b_1} + g_{b_2}\big(r_{b_2} + \cdots + g_{b_{K-1}}\big(r_{b_{K-1}} + g_{b_K} r_{b_K}\big)\big), \end{aligned} \tag{12}$$

where $\mathbb{1}(\cdot)$ is an indicator function and $\alpha_{b_j} (j > 1)$ is a layer-wise threshold that controls the choice of bitwidth. Specifically, if the quantization error $||z - z_{b_{j-1}}||$ is greater than α_{b_j}, we activate the corresponding quantization gate to increase the bitwidth so that the residual error can be reduced, and vice versa.

Note that in Eq. (12), we can also consider filter pruning as 0-bit filter-wise quantization. To avoid the prohibitively large filter-wise search space, we propose to divide the filters into groups based on channel indices and consider the group-wise sparsity instead. To be specific, we introduce a binary gate g_{c,b_1} for each group to encode the choice of pruning as:

$$\begin{aligned} g_{c,b_1} &= \mathbb{1}(||w_c|| - \alpha_{b_1} > 0), \\ z_{c,b_K} &= g_{c,b_1} \cdot \big(z_{c,b_1} + g_{b_2}\big(r_{c,b_2} + \cdots \\ &\quad + g_{b_{K-1}}\big(r_{c,b_{K-1}} + g_{b_K} r_{c,b_K}\big)\big)\big), \end{aligned} \tag{13}$$

where z_{c,b_K} is the c-th group of quantized filters and r_{c,b_j} is the corresponding re-assignment offset by quantizing the residual error $z_c - z_{c,b_{j-1}}$. Here, α_{b_1} is a layer-wise threshold for filter pruning. Following PFEC (Li et al., 2017a), we use the ℓ_1-norm to evaluate the importance of the filter. Specifically, if a group of filters is important, the corresponding pruning gate will be activated and vice versa.

Search space for model compression. Given an uncompressed network with L layers, we use C_l to denote the number of filters at the l-th layer. Let B be the number of filters in a group. For any layer l, there would be $G = \lceil \frac{C_l}{B} \rceil$ groups in total. Since we quantize both weights and activations, given K candidate bitwidths, there are K^2 different quantization configurations for each layer. Thus, for the whole network with L layers, the size of the search space Ω can be computed by

$$|\Omega| = \prod_{l=1}^{L} \left(K^2 \times G\right). \tag{14}$$

Eq. (14) indicates that the search space is large enough to cover the potentially good configurations.

4.3. Learning for Loss-aware Compression

Note that both quantization and pruning have their corresponding thresholds. To determine the thresholds, we propose to learn them via gradient descent. Nevertheless, the indicator function in Eqs. (12) and (13) is non-differentiable. To address this, we use straight-through estimator (STE) (Bengio et al., 2013; Zhou et al., 2016) to approximate the gradient of the indicator function $\mathbb{1}(\cdot)$ using the gradient of the sigmoid function $S(\cdot)$:

$$\begin{aligned} \frac{\partial g}{\partial \alpha} = \frac{\partial \mathbb{1}\left(A - \alpha > 0\right)}{\partial \alpha} &\approx \frac{\partial S\left(A - \alpha\right)}{\partial \alpha} \\ &= -S\left(A - \alpha\right)\left(1 - S(A - \alpha)\right), \end{aligned} \tag{15}$$

where g is the output of a binary gate, α is the corresponding threshold and A denotes some metrics (*i.e.*, quantization error $||z - z_{b_{j-1}}||$ or the ℓ_1-norm of the filter $||w_c||$).

Training objective. To design a hardware-efficient model, the objective should reflect both the accuracy and computational cost of a compressed model. Following (Cai et al., 2019b), we incorporate the computational cost into the objective function and formulate the joint objective as:

$$\mathcal{L}(\mathbf{W}, \alpha) = \mathcal{L}_{ce}(\mathbf{W}, \alpha) + \lambda \log R(\mathbf{W}, \alpha), \tag{16}$$

where $\mathcal{L}_{ce}(\cdot)$ is the cross-entropy loss, $R(\cdot)$ is the computational cost of the network and λ is a balancing hyperparameter. Here, we can use the Bit-Operation (BOP) count (Guo et al., 2020; Ying et al., 2020) or inference latency to measure the computational cost. Following single-path NAS (Stamoulis et al., 2019), we use a similar formulation of computational cost to preserve the differentiability of the objective function. The details of the differentiable computational loss are put in the supplementary material.

By jointly training the binary gates and model parameters, the pruning rate and bitwidth of each layer can be automatically determined. However, the gradient approximation of the binary gate inevitably introduces noisy signals, which can be even more severe when we quantize both weights and activations. Thus, we propose to train the binary gates of weights and activations in an alternative manner. The details training method are put in the supplementary material. Once the training is finished, we can obtain the compressed model by selecting those filters and bitwidths with activated binary gates. Then, we fine-tune the compressed model to compensate for the accuracy loss.

More discussions. Our single-path scheme is different from the multi-path scheme in DNAS (Wu et al., 2018). To study the difference, for simplicity, we consider a linear regression scenario and compare the quantization errors incurred by the two quantization schemes over the regression weights. Formally, we have the following results on the quantization errors incurred by the two schemes.

Proposition 1 *Consider a linear regression problem* $\min_{\mathbf{w}\in\mathbb{R}^d} \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}}[(y-\mathbf{w}\mathbf{x})^2]$ *with data pairs* $\{(\mathbf{x},y)\}$*, where* $\mathbf{x}\in\mathbb{R}^d$ *is sampled from* $\mathcal{N}(0,\sigma^2\mathbf{I})$ *and* $y\in\mathbb{R}$ *is its response. Consider using LBS and DNAS to quantize the linear regression weights. Let* $\mathbf{w}_L^t$ *and* $\mathbf{w}_D^t$ *be the quantized regression weights of LBS and DNAS at the iteration* t *of the optimization, respectively. Then the following equivalence holds during the optimization process:*

$$\lim_{t\to\infty} \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}}[(y-\mathbf{w}_L^t\mathbf{x})^2] = \lim_{t\to\infty} \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}}[(y-\mathbf{w}_D^t\mathbf{x})^2],$$
$$\text{where } \mathbf{w}_L^t = \mathbf{w}_{b_1}^t + g_{b_2}^t(\mathbf{r}_{b_2}^t + \cdots + g_{b_{K-1}}^t(\mathbf{r}_{b_{K-1}}^t + g_{b_K}^t\mathbf{r}_{b_K}^t)),$$
$$\mathbf{r}_{b_j}^t = D(\mathbf{w}_L^t - \mathbf{w}_{L,b_{j-1}}^t, s_{b_j}),\ j=2,\cdots,K,$$
$$\mathbf{w}_D^t = \sum_{i=1}^K p_i^t \mathbf{w}_{b_i}^t, \quad \sum_{i=1}^K p_i^t = 1. \tag{17}$$

From Proposition 1, the multi-path scheme (DNAS) converges to our single-path scheme during the optimization. However, our single-path scheme contains fewer parameters to learn and thus has a tight error bound (See also in Corollary 1). As shown in Figure 2, our method also converges faster and smoother empirically than the multi-path scheme.

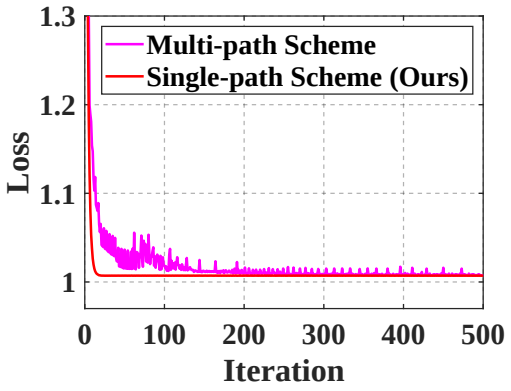


Figure 2. Comparison between our single-path scheme (Ours) and multi-path scheme (Wu et al., 2018). The experimental settings for generating the figure are put in the supplementary material.

5. Experiments

Compared methods. To investigate the effectiveness of LBS, we consider the following methods for comparisons: **LBS**: our proposed method with joint pruning and quantization; **LBS-Q**: LBS with quantization only; **LBS-P**: LBS with pruning only; and several state-of-the-art model compression methods including DNAS (Wu et al., 2018), HAQ (Wang et al., 2019), DQ (Uhlich et al., 2020), Bayesian Bits (van Baalen et al., 2020), and DJPQ (Ying et al., 2020).

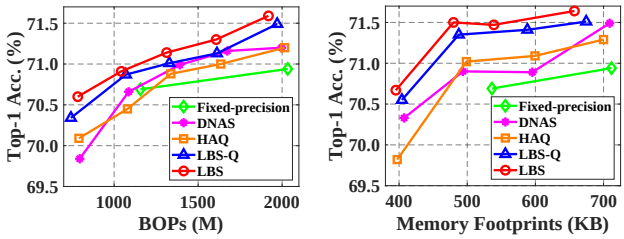
Evaluation metrics. We measure the performance of different methods in terms of the Top-1 and Top-5 accuracy. Experiments on CIFAR-100 are repeated 5 times and we report the mean and standard deviation. Following (Guo et al., 2020; Ying et al., 2020), we measure the computational cost by the Bit-Operation (BOP) count. The BOP compression ratio is defined as the ratio between the total BOPs of the uncompressed and compressed models. We can also measure the computational cost with the total weights and activations memory footprints following DQ (Uhlich et al., 2020). Moreover, following (Stamoulis et al., 2019; Liu et al., 2019a), we use the search cost to measure the time of finding an optimal compressed model.

Implementation details. Following HAQ (Wang et al., 2019), we quantize all the layers, in which the first and the last layers are quantized to 8-bit. Following ThiNet (Luo et al., 2017), we only conduct filter pruning for the first layer in the residual block. For ResNet-20 and ResNet-56 on CIFAR-100 (Krizhevsky et al., 2009), we set B to 4. For ResNet-18 and MobileNetV2 on ImageNet (Russakovsky et al., 2015), B is set to 16 and 8, respectively. We first train the full-precision models and then use the pre-trained weights to initialize the compressed models. More details can be found in the supplementary material.

Note that in Theorem 1, both the smallest bitwidth b_1 and the quantization factor γ_j can be set to arbitrary appropri-

Table 1. Comparisons of different methods on CIFAR-100.

Network	Method	BOPs (M)	BOP comp. ratio	Top-1 Acc. (%)	Top-5 Acc. (%)
ResNet-20	Full-precision	41798.6	1.0	67.5	90.8
	4-bit precision	674.6	62.0	67.8±0.3	90.4±0.2
	DQ	1180.0	35.4	67.7±0.6	90.4±0.5
	HAQ	653.4	64.0	67.7±0.1	90.4±0.3
	DNAS	660.0	62.9	67.8±0.3	90.4±0.2
	LBS-P (Ours)	28586.5	1.5	67.9±0.1	90.7±0.2
	LBS-Q (Ours)	649.5	64.4	68.1±0.1	90.5±0.0
	LBS (Ours)	**630.6**	**66.3**	**68.1±0.3**	**90.6±0.2**
ResNet-56	Full-precision	128771.7	1.0	71.7	92.2
	4-bit precision	2033.6	63.3	70.9±0.3	91.2±0.4
	DQ	2222.9	57.9	70.7±0.2	91.4±0.4
	HAQ	2014.9	63.9	71.2±0.1	91.1±0.2
	DNAS	2035.7	65.3	71.2±0.2	91.3±0.3
	LBS-P (Ours)	87021.6	1.5	71.5±0.1	91.8±0.2
	LBS-Q (Ours)	1970.7	65.3	71.5±0.2	91.5±0.2
	LBS (Ours)	**1918.8**	**67.1**	**71.6±0.1**	**91.8±0.4**



(a) BOPs vs. Top-1 Acc. (b) Memory vs. Top-1 Acc.

Figure 3. Results of different compressed networks with different BOPs and memory footprints. We use different methods to compress ResNet-56 and report the results on CIFAR-100.

ate integer values (*e.g.*, 2, 3, etc.). To obtain a hardware-friendly compressed network[1], we set b_1 and γ_j to 2, which ensures that all the decomposition bitwidths are power-of-two. Moreover, since the quantization error of 8-bit values is small enough, we only consider those bitwidths that are not greater than 8-bit (*i.e.*, $b_j \in \{2, 4, 8\}$).

5.1. Main Results

We apply the proposed method to compress ResNet-20 and ResNet-56 on CIFAR-100 and ResNet-18 and MobileNetV2 on ImageNet. We compare the performance of different methods in Table 1 and Table 2. We also show the results of the compressed ResNet-56 with different BOPs and memory footprints in Figure 3. From the results, 4-bit quantized networks of ResNet achieve lossless performance. Also, the 8-bit MobileNetV2 only leads to a 0.4% performance decline in terms of the Top-1 Accuracy. Compared with fixed-precision quantization, mixed-precision methods are able to reduce the BOPs while preserving the performance. Notably, our proposed LBS-Q outperforms the state-of-the-art baselines with less computational cost. Specifically, LBS-Q compressed ResNet-18 outperforms the one compressed by HAQ with more BOPs reduction. More critically, our proposed LBS achieves significant improvement in terms of BOPs and memory footprints. For example, in Figure 3(b), our LBS compressed ResNet-56 model yields much fewer memory footprints (395.25 vs. 536.24) but achieves comparable performance compared with the fixed-precision counterpart. Moreover, by combining pruning and quantization, LBS achieves nearly lossless performance while further reducing the computational cost of LBS-Q. For example, LBS compressed ResNet-18 reduces the BOPs by 57.5× while still outperforming the full-precision model by 0.1% in terms of the Top-1 accuracy on ImageNet.

[1]More details can be found in the supplementary material.

5.2. Further Studies

Effect of the bit sharing scheme. To investigate the effect of the bit sharing scheme, we apply LBS to quantize ResNet-20 and ResNet-56 with and without the bit sharing scheme on CIFAR-100. Here, LBS without the bit sharing denotes that we compress the models with multi-path scheme (Wu et al., 2018). We report the testing accuracy and BOPs in Table 3. We also present the search cost and GPU memory footprint measured on a GPU device (NVIDIA TITAN Xp). From the results, LBS with the bit sharing scheme consistently outperforms the ones without the bit sharing scheme while significantly reducing the search cost and GPU memory footprint.

Effect of the one-stage compression. To investigate the effect of the one-stage compression scheme (perform pruning and quantization jointly), we extend LBS to two-stage optimization, where we sequentially perform filter pruning and quantization, denoted as LBS-P→LBS-Q for conve-

Table 2. Comparisons on ImageNet. "*" denotes that we get the results from the figures in (van Baalen et al., 2020) and "–" denotes that the results are not reported.

Network	Method	BOPs (G)	BOP comp. ratio	Top-1 Acc. (%)	Top-5 Acc. (%)
ResNet-18	Full-precision	1857.6	1.0	70.7	89.8
	4-bit precision	34.7	53.5	71.0	89.8
	DQ	40.7	40.6	68.5	–
	DJPQ	35.5	52.3	69.1	–
	HAQ	34.7	53.5	70.2	89.5
	Bayesian Bits*	35.9	51.7	69.5	–
	LBS-Q (Ours)	33.1	56.1	**70.9**	**89.7**
	LBS (Ours)	**32.3**	**57.5**	70.8	89.6
MobileNetV2	Full-precision	308.0	1.0	71.9	90.3
	8-bit precision	19.2	16.0	71.5	90.1
	DQ	19.6	1.9	70.4	89.7
	HAQ	13.8	22.3	71.4	90.2
	Bayesian Bits*	13.8	22.3	71.4	–
	LBS-Q (Ours)	13.6	22.6	71.6	90.2
	LBS (Ours)	**13.6**	**22.6**	**71.8**	**90.3**

Table 3. Effect of the bit sharing scheme. We report the testing accuracy, BOPs, search cost, and GPU memory footprint on CIFAR-100. The search cost is measured on a GPU device (NVIDIA TITAN Xp).

Network	Method	Top-1 Acc.	Top-5 Acc.	BOPs (M)	Search Cost (GPU hours)	GPU Memory (GB)
ResNet-20	w/o bit sharing	67.8±0.1	90.5±0.2	664.2	2.8	4.4
	w/ bit sharing	**68.1±0.1**	**90.5±0.0**	**649.5**	**0.8**	**1.5**
ResNet-56	w/o bit sharing	71.3±0.3	91.4±0.4	2001.1	8.7	10.9
	w/ bit sharing	**71.5±0.2**	**91.5±0.2**	**1970.7**	**1.9**	**3.0**

Table 4. Effect of the one-stage compression. We report the results of ResNet-56 on CIFAR-100.

Network	Method	Top-1 Acc.	Top-5 Acc.	BOPs (M)
ResNet-56	LBS-P → LBS-Q	70.4±0.1	90.8±0.2	1077.7
	LBS	**70.8±0.4**	**91.2±0.1**	**1042.5**

nience. The results are shown in Table 4. Compared with the two-stage counterpart, LBS achieves better performance with less computational cost, which shows the superiority of the one-stage compression.

Table 5. Effect of the alternative training scheme. We report the results of ResNet-56 on CIFAR-100.

Network	Method	Top-1 Acc.	Top-5 Acc.	BOPs (M)
ResNet-56	Joint	71.3±0.2	91.6±0.3	1942.4
	Alternative	**71.6±0.1**	**91.8±0.4**	**1918.8**

Effect of the alternative training scheme. To investigate the effect of the alternative training scheme introduced in Section 4.3, we apply our method to compress ResNet-56 using a joint training scheme and an alternative training scheme on CIFAR-100. Here, the joint training scheme denotes that we train the binary gates of weights and activations jointly. From the results of Table 5, the model trained with the alternative scheme achieves better performance than the joint one, which demonstrates the effectiveness of the alternative training scheme.

6. Conclusion and Future Work

In this paper, we have proposed a novel model compression method called Loss-aware Bit Sharing (LBS). The proposed LBS introduces a novel single-path bit sharing model, which encodes all bitwidths in the search space. Specifically, our LBS is based on the observation that the quantized values of a high bitwidth are shared with those of lower bitwidths under some constraints. Therefore, we have proposed the single-path bit sharing decomposition that decomposes a quantized representation into the sum of the lowest bitwidth representation and a series of re-assignment offsets. Based on this, we have further introduced learnable binary gates to encode the choice of different compression configurations. By training the binary gates, the optimal compression ratio of each layer can be automatically determined. Experiments on CIFAR-100 and ImageNet have shown that our method is able to achieve significant cost reduction while preserving performance. In the future, we plan to work on a joint search for architecture, pruning, and quantization to find a compact model with better performance.

References

Bai, Y., Wang, Y.-X., and Liberty, E. Proxquant: Quantized neural networks via proximal operators. In *Proc. Int. Conf. Learn. Repren.*, 2019.

Bengio, Y., Léonard, N., and Courville, A. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

Bhalgat, Y., Lee, J., Nagel, M., Blankevoort, T., and Kwak, N. Lsq+: Improving low-bit quantization through learnable offsets and better initialization. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2020.

Burgess, J. Rtx on – the nvidia turing gpu. In *2019 IEEE Hot Chips 31 Symposium (HCS)*, pp. 1–27, 2019.

Cai, H., Zhu, L., and Han, S. Proxylessnas: Direct neural architecture search on target task and hardware. In *Proc. Int. Conf. Learn. Repren.*, 2019a.

Cai, H., Zhu, L., and Han, S. ProxylessNAS: Direct neural architecture search on target task and hardware. In *Proc. Int. Conf. Learn. Repren.*, 2019b.

Cai, Z. and Vasconcelos, N. Rethinking differentiable search for mixed-precision neural networks. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2020.

Cai, Z., He, X., Sun, J., and Vasconcelos, N. Deep learning with low precision by half-wave gaussian quantization. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2017.

Chen, Y., Guan, T., and Wang, C. Approximate nearest neighbor search by residual vector quantization. *Sensors*, 10(12):11259–11273, 2010.

Choi, J., Wang, Z., Venkataramani, S., Chuang, P. I.-J., Srinivasan, V., and Gopalakrishnan, K. Pact: Parameterized clipping activation for quantized neural networks. *arXiv preprint arXiv:1805.06085*, 2018.

Ding, R., Chin, T.-W., Liu, Z., and Marculescu, D. Regularizing activation distribution for training binarized deep networks. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2019.

Dong, X. and Yang, Y. Network pruning via transformable architecture search. In *Proc. Adv. Neural Inf. Process. Syst.*, 2019.

Dong, Z., Yao, Z., Gholami, A., Mahoney, M. W., and Keutzer, K. Hawq: Hessian aware quantization of neural networks with mixed-precision. In *Proc. IEEE Int. Conf. Comp. Vis.*, 2019.

Esser, S. K., McKinstry, J. L., Bablani, D., Appuswamy, R., and Modha, D. S. Learned step size quantization. In *Proc. Int. Conf. Learn. Repren.*, 2020.

Gong, Y., Liu, L., Yang, M., and Bourdev, L. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*, 2014.

Guo, Y., Yao, A., and Chen, Y. Dynamic network surgery for efficient dnns. In *Proc. Adv. Neural Inf. Process. Syst.*, 2016.

Guo, Y., Zheng, Y., Tan, M., Chen, Q., Chen, J., Zhao, P., and Huang, J. Nat: Neural architecture transformer for accurate and compact architectures. In *Proc. Adv. Neural Inf. Process. Syst.*, 2019.

Guo, Z., Zhang, X., Mu, H., Heng, W., Liu, Z., Wei, Y., and Sun, J. Single path one-shot neural architecture search with uniform sampling. In *Proc. Eur. Conf. Comp. Vis.*, 2020.

Han, S., Mao, H., and Dally, W. J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *Proc. Int. Conf. Learn. Repren.*, 2016.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2016.

He, Y., Zhang, X., and Sun, J. Channel pruning for accelerating very deep neural networks. In *Proc. IEEE Int. Conf. Comp. Vis.*, 2017.

He, Y., Lin, J., Liu, Z., Wang, H., Li, L.-J., and Han, S. Amc: Automl for model compression and acceleration on mobile devices. In *Proc. Eur. Conf. Comp. Vis.*, 2018.

He, Y., Liu, P., Wang, Z., Hu, Z., and Yang, Y. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2019.

Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., and Bengio, Y. Binarized neural networks. In *Proc. Adv. Neural Inf. Process. Syst.*, 2016.

Jin, Q., Yang, L., and Liao, Z. Towards efficient training for neural network quantization. *arXiv preprint arXiv:1912.10207*, 2019.

Jung, S., Son, C., Lee, S., Son, J., Han, J.-J., Kwak, Y., Hwang, S. J., and Choi, C. Learning to quantize deep networks by optimizing quantization intervals with task loss. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2019.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. 2009.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Proc. Adv. Neural Inf. Process. Syst.*, 2012.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

Li, H., Kadav, A., Durdanovic, I., Samet, H., and Graf, H. P. Pruning filters for efficient convnets. In *Proc. Int. Conf. Learn. Repren.*, 2017a.

Li, Y., Dong, X., and Wang, W. Additive powers-of-two quantization: An efficient non-uniform discretization for neural networks. In *Proc. Int. Conf. Learn. Repren.*, 2020.

Li, Z., Ni, B., Zhang, W., Yang, X., and Gao, W. Performance guaranteed network acceleration via high-order residual quantization. In *Proc. IEEE Int. Conf. Comp. Vis.*, 2017b.

Lin, T.-Y., Dollár, P., Girshick, R., He, K., Hariharan, B., and Belongie, S. Feature pyramid networks for object detection. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2017a.

Lin, T.-Y., Goyal, P., Girshick, R., He, K., and Dollár, P. Focal loss for dense object detection. In *Proc. IEEE Int. Conf. Comp. Vis.*, 2017b.

Liu, H., Simonyan, K., and Yang, Y. DARTS: Differentiable architecture search. In *Proc. Int. Conf. Learn. Repren.*, 2019a.

Liu, Z., Mu, H., Zhang, X., Guo, Z., Yang, X., Cheng, K.-T., and Sun, J. Metapruning: Meta learning for automatic neural network channel pruning. In *Proc. IEEE Int. Conf. Comp. Vis.*, 2019b.

Loshchilov, I. and Hutter, F. SGDR: stochastic gradient descent with warm restarts. In *Proc. Int. Conf. Learn. Repren.*, 2017.

Lou, Q., Liu, L., Kim, M., and Jiang, L. Autoqb: Automl for network quantization and binarization on mobile devices. *arXiv preprint arXiv:1902.05690*, 2019.

Louizos, C., Reisser, M., Blankevoort, T., Gavves, E., and Welling, M. Relaxed quantization for discretized neural networks. In *Proc. Int. Conf. Learn. Repren.*, 2019.

Luo, J.-H., Wu, J., and Lin, W. Thinet: A filter level pruning method for deep neural network compression. In *Proc. IEEE Int. Conf. Comp. Vis.*, 2017.

Nesterov, Y. E. A method for solving the convex programming problem with convergence rate o (1/kˆ 2). In *Proceedings of the USSR Academy of Sciences*, volume 269, pp. 543–547, 1983.

Pham, H., Guan, M., Zoph, B., Le, Q., and Dean, J. Efficient neural architecture search via parameter sharing. In *Proc. Int. Conf. Mach. Learn.*, 2018.

Real, E., Aggarwal, A., Huang, Y., and Le, Q. V. Regularized evolution for image classifier architecture search. In *Proc. AAAI Conf. on Arti. Intel.*, 2019.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. Imagenet large scale visual recognition challenge. *Int. J. Comp. Vis.*, 115(3):211–252, 2015.

Sharma, H., Park, J., Suda, N., Lai, L., Chau, B., Chandra, V., and Esmaeilzadeh, H. Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network. In *International Symposium on Computer Architecture*, pp. 764–775. IEEE, 2018.

Stamoulis, D., Ding, R., Wang, D., Lymberopoulos, D., Priyantha, B., Liu, J., and Marculescu, D. Single-path nas: Designing hardware-efficient convnets in less than 4 hours. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 481–497, 2019.

Tung, F. and Mori, G. Clip-q: Deep network compression learning by in-parallel pruning-quantization. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pp. 7873–7882, 2018.

Uhlich, S., Mauch, L., Cardinaux, F., Yoshiyama, K., Garcia, J. A., Tiedemann, S., Kemp, T., and Nakamura, A. Mixed precision dnns: All you need is a good parametrization. In *Proc. Int. Conf. Learn. Repren.*, 2020.

van Baalen, M., Louizos, C., Nagel, M., Amjad, R. A., Wang, Y., Blankevoort, T., and Welling, M. Bayesian bits: Unifying quantization and pruning. In *Proc. Adv. Neural Inf. Process. Syst.*, 2020.

Wang, K., Liu, Z., Lin, Y., Lin, J., and Han, S. Haq: Hardware-aware automated quantization with mixed precision. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2019.

Wang, T., Wang, K., Cai, H., Lin, J., Liu, Z., Wang, H., Lin, Y., and Han, S. Apq: Joint search for network architecture, pruning and quantization policy. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2020.

Wu, B., Wang, Y., Zhang, P., Tian, Y., Vajda, P., and Keutzer, K. Mixed precision quantization of convnets via differentiable neural architecture search. *arXiv preprint arXiv:1812.00090*, 2018.

Yang, H., Gui, S., Zhu, Y., and Liu, J. Automatic neural network compression by sparsity-quantization joint learning: A constrained optimization-based approach. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2020.

Ye, S., Zhang, T., Zhang, K., Li, J., Xie, J., Liang, Y., Liu, S., Lin, X., and Wang, Y. A unified framework of dnn weight pruning and weight clustering/quantization using admm. In *Proc. AAAI Conf. on Arti. Intel.*, 2019.

Ying, W., Yadong, L., and Tijmen, B. Differentiable joint pruning and quantization for hardware efficiency. In *Proc. Eur. Conf. Comp. Vis.*, 2020.

Zhang, D., Yang, J., Ye, D., and Hua, G. Lq-nets: Learned quantization for highly accurate and compact deep neural networks. In *Proc. Eur. Conf. Comp. Vis.*, 2018.

Zhou, S., Wu, Y., Ni, Z., Zhou, X., Wen, H., and Zou, Y. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.

Zhuang, B., Shen, C., Tan, M., Liu, L., and Reid, I. Towards effective low-bitwidth convolutional neural networks. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2018a.

Zhuang, B., Shen, C., Tan, M., Liu, L., and Reid, I. Structured binary neural networks for accurate image classification and semantic segmentation. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2019.

Zhuang, B., Liu, L., Tan, M., Shen, C., and Reid, I. Training quantized neural networks with a full-precision auxiliary module. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, 2020.

Zhuang, Z., Tan, M., Zhuang, B., Liu, J., Guo, Y., Wu, Q., Huang, J., and Zhu, J. Discrimination-aware channel pruning for deep neural networks. In *Proc. Adv. Neural Inf. Process. Syst.*, 2018b.

7. Appendix

In the supplementary, we provide the detailed proof of Theorem 7.1, Corollary 7.2, Proposition 1, more details and more experimental results of the proposed LBS. We organize the supplementary material as follows.

- In Section 7.1, we provide the proof for Theorem 1.
- In Section 7.2, we give the proof for Corollary 1.
- In Section 7.3, we offer more details about the proof of Proposition 1.
- In Section 7.4, we provide more details about the differentiable computational cost.
- In Section 7.5, we introduce the training algorithm for LBS.
- In Section 7.6, we show more experimental details about LBS.
- In Section 7.7, we provide more explanations on the hardware-friendly decomposition.
- In Section 7.8, we describe the details about the quantization configurations of different methods.
- In Section 7.9, we give more results on resource-constrained compression.
- In Section 7.10, we provide more comparisons in terms of search efficiency.
- In Section 7.11, we give more comparisons in terms of memory footprints.
- In Section 7.12, we offer more results in terms of MobileNetV3 on CIFAR-100.
- In Section 7.13, we report the detailed about compression configurations of the compressed models.

7.1. Proof of Theorem 1

To prove Theorem 1, we first provide a lemma of quantization formulas as follows.

Lemma 1 *Let $z \in [0,1]$ be a normalized full-precision input, and $\{b_j\}_{j=1}^{K}$ be a sequence of candidate bitwidths. If the bitwidth b_j is an integer multiple of b_{j-1},* i.e.*, $b_j{=}\gamma_j b_{j-1}(j>1)$, where $\gamma_j \in \mathbb{Z}^+\backslash\{1\}$ is a multiplier, then the quantized $z_{b_{j+1}}$ can be decomposed into the quantized z_{b_j} and quantized value re-assignment $r_{b_{j+1}}$*

$$\begin{aligned} z_{b_{j+1}} &= z_{b_j} + r_{b_{j+1}}, \\ \text{where } r_{b_{j+1}} &= D(z - z_{b_j}, s_{b_{j+1}}), \\ z_{b_j} &= D(z, s_{b_j}), \\ s_{b_j} &= \frac{1}{2^{b_j}-1}. \end{aligned} \tag{18}$$

Proof: We construct two sequences $\{A_m\}$ and $\{B_n\}$ for the b_j-quantized value and the b_{j+1}-quantized value

$$\left\{0, \frac{1}{2^{b_j}-1}, \frac{2}{2^{b_j}-1}, \ldots, \frac{2^{b_j}-2}{2^{b_j}-1}, 1\right\}, \tag{19}$$

$$\left\{0, \frac{1}{2^{b_{j+1}}-1}, \frac{2}{2^{b_{j+1}}-1}, \ldots, \frac{2^{b_{j+1}}-2}{2^{b_{j+1}}-1}, 1\right\}. \tag{20}$$

First, we can obtain each value in $\{A_m\}$ is in $\{B_n\}$, *i.e.*, $\{A_m\} \subset \{B_n\}$ since $2^{b_{j+1}}{-}1{=}(2^{b_j})^{\gamma_{j+1}}{-}1$ is divisible by $2^{b_j}{-}1$. Then we rewrite the sequence $\{B_n\}$ as:

$$\left\{0, \frac{1}{2^{b_{j+1}}-1}, \ldots, \underbrace{\frac{1}{2^{b_j}-1}}_{A_2}, \ldots, \underbrace{\frac{t-1}{2^{b_{j+1}}-1}}_{B_t}, \underbrace{\frac{t}{2^{b_{j+1}}-1}}_{B_{t+1}}, \ldots, \underbrace{\frac{2}{2^{b_j}-1}}_{A_3}, \ldots, \frac{2^{b_{j+1}}-2}{2^{b_{j+1}}-1}, 1\right\}. \tag{21}$$

Note that $A_1, A_2, \ldots \in \{B_n\}$, thus we only focus on the sequence B_n. For an attribute $z \in [0,1]$, it surely falls in some interval $[A_i, A_{i+1}]$. Without loss of generality, we assume $z \in [A_2, A_3]$ and $z \in [B_t, B_{t+1}]$ (see sequence 21).

Next, we will discuss the following cases according to the position of z in $[A_2, A_3]$:
1) If $z \in [A_2, \frac{A_2+A_3}{2}]$, based on the definition of $z_{b_j} = D(z, s_{b_j})$, we have $z_{b_j} = A_2$. Moreover, if $z \in [B_t, \frac{B_t+B_{t+1}}{2}]$, then $z_{b_{j+1}} = B_t$ and $r_{b_{j+1}} = D(z-A_2, s_{b_{j+1}}) = B_t - A_2$. Thus, we get $z_{b_{j+1}} = z_{b_j} + r_{b_{j+1}}$; otherwise if $z \in (\frac{B_t+B_{t+1}}{2}, B_{t+1}]$, then $z_{b_{j+1}} = B_{t+1}$ and $r_{b_{j+1}} = B_{t+1} - A_2$. Therefore, we have the same conclusion.
2) If $z \in (\frac{A_2+A_3}{2}, A_3]$, similar to the first case, we have $z_{b_j} = A_3$. Moreover, if $z \in [B_t, \frac{B_t+B_{t+1}}{2}]$, then $z_{b_{j+1}} = B_t$ and $r_{b_{j+1}} = D(z - A_3, s_{b_{j+1}}) = -(A_3 - B_t)$. Thus we have $z_{b_{j+1}} = z_{b_j} + r_{b_{j+1}}$; otherwise if $z \in (\frac{B_t+B_{t+1}}{2}, B_{t+1}]$, then $z_{b_{j+1}} = B_{t+1}$ and $r_{b_{j+1}} = -(A_3 - B_{t+1})$. Hence we still obtain the same conclusion.

□

Theorem 2 *Let $z \in [0,1]$ be a normalized full-precision input, and $\{b_j\}_{j=1}^K$ be a sequence of candidate bitwidths. If the bitwidth b_j is an integer multiple of b_{j-1},* i.e., *$b_j = \gamma_j b_{j-1} (j > 1)$, where $\gamma_j \in \mathbb{Z}^+ \backslash \{1\}$ is a multiplier, then the following quantized approximation z_{b_K} can be decomposed as:*

$$\begin{aligned} z_{b_K} &= z_{b_1} + \sum_{j=2}^{K} r_{b_j}, \\ \text{where } r_{b_j} &= D(z - z_{b_{j-1}}, s_{b_j}), \\ z_{b_j} &= D(z, s_{b_j}), \\ s_{b_j} &= \frac{1}{2^{b_j} - 1}. \end{aligned} \tag{22}$$

Proof: By summing the equations $z_{b_j} = z_{b_{j-1}} + r_{b_j}$ in lemma 1 from $j=2$ to K, we complete the results. □

7.2. Proof of Corollary 1

Corollary 2 (Normalized Quantization Error Bound) *Given $\mathbf{z} \in [0,1]^d$ being a normalized full-precision vector, $\mathbf{z}_{b_K}$ being its quantized vector with bitwidth b_K, where d is the cardinality of $\mathbf{z}$, let $\epsilon_K = \frac{\|\mathbf{z} - \mathbf{z}_{b_K}\|_1}{\|\mathbf{z}\|_1}$ be the normalized quantization error, then the following error bound w.r.t. K holds*

$$|\epsilon_K - \epsilon_{K+1}| \leq \frac{C}{2^{b_K} - 1}, \tag{23}$$

where $C = \frac{d}{\|\mathbf{z}\|_1}$ is a constant.

Proof: Let $\mathbf{z} = [z^1, z^2, \ldots, z^d]$ and $\mathbf{z}_{b_K} = [z^1_{b_K}, z^2_{b_K}, \ldots, z^d_{b_K}]$, we rewrite the normalized quantization error as:

$$\epsilon_K = \frac{\sum_{i=1}^{d} |z^i - z^i_{b_K}|}{\|\mathbf{z}\|_1}. \tag{24}$$

Then, we get the following result

$$\begin{aligned} |\epsilon_K - \epsilon_{K+1}| &= \frac{1}{\|\mathbf{z}\|_1} \sum_{i=1}^{d} \Big| |z^i - z^i_{b_K}| - |z^i - z^i_{b_{K+1}}| \Big| \leq \frac{1}{\|\mathbf{z}\|_1} \sum_{i=1}^{d} \left| z^i_{b_K} - z^i_{b_{K+1}} \right| = \frac{1}{\|\mathbf{z}\|_1} \sum_{i=1}^{d} \left| r^i_{b_{K+1}} \right| \\ &\leq \frac{1}{\|\mathbf{z}\|_1} \sum_{i=1}^{d} |s_{b_K}| = \frac{d}{\|\mathbf{z}\|_1 (2^{b_K} - 1)}. \end{aligned} \tag{25}$$

□

To empirically demonstrate Corollary 1, we perform quantization on a random toy data $\mathbf{z} \in [0,1]^{100}$ and show the normalized quantization error change $|\epsilon_K - \epsilon_{K+1}|$ and error bound $\frac{C}{2^{b_K}-1}$ in Figure 4. From the results, the normalized quantization error change (red line) is bounded by the error bound (blue line).

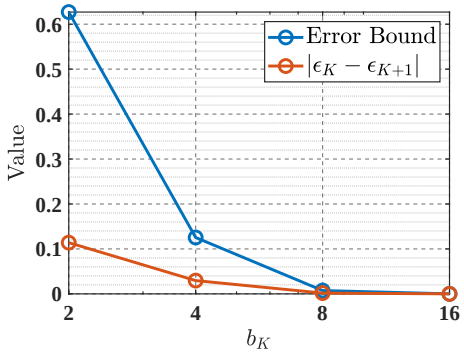


Figure 4. Two types of errors vs. Bitwidth. The red line denotes the curve of the normalized quantization error change $|\epsilon_K - \epsilon_{K+1}|$ and the blue line is the curve of error bound $\frac{C}{2^{b_K}-1}$.

7.3. Proof of Proposition 1

To prove Proposition 1, we derive a lemma of normalized quantization error as follows.

Lemma 2 *Given $\mathbf{z} \in [0,1]^d$ being a normalized full-precision vector and $\mathbf{z}_{b_K}$ being its quantized vector with bitwidth b_K, where d is the cardinality of $\mathbf{z}$, let $\epsilon_K = \frac{\|\mathbf{z}-\mathbf{z}_{b_K}\|_1}{\|\mathbf{z}\|_1}$ be the normalized quantization error, then ϵ_K decreases with the increase of K.*

Proof: Let $\mathbf{z} = [z^1, z^2, \dots, z^d]$ and $\mathbf{z}_{b_K} = [z^1_{b_K}, z^2_{b_K}, \dots, z^d_{b_K}]$, we rewrite the normalized quantization error as:

$$\epsilon_K = \frac{\sum_{i=1}^d |z^i - z^i_{b_K}|}{\|\mathbf{z}\|_1}. \tag{26}$$

Since z_{b_K} is the only variable, we focus on one term $z - z_{b_K}$ of the numerator in Eq. (26). Next, we will prove $|z - z_{b_j}| \geq |z - z_{b_{j+1}}|$. Following the sequence (21) in Lemma 1 and without loss of generality, we assume $z \in [B_t, B_{t+1}] \subset [A_2, A_3]$, then we have the following conclusions:
1) If $A_2 = B_t < B_{t+1} < A_3$, then $|z - z_{b_j}| = |z - A_2| \geq |z - z_{b_{j+1}}|$, where $|z - A_2| = |z - z_{b_{j+1}}|$ happens only when $z \in [B_t, \frac{B_t + B_{t+1}}{2}]$;
2) If $A_2 < B_t < B_{t+1} < A_3$, then $|z - z_{b_j}| \geq B_{t+1} - B_t > \frac{B_{t+1} - B_t}{2} \geq |z - z_{b_{j+1}}|$;
3) If $A_2 < B_t < B_{t+1} = A_3$, then $|z - z_{b_j}| = |z - A_3| \geq |z - z_{b_{j+1}}|$, where $|z - A_3| = |z - z_{b_{j+1}}|$ happens only when $z \in [\frac{B_t + B_{t+1}}{2}, B_{t+1}]$.

Therefore, we have $|z - z_{b_j}| \geq |z - z_{b_{j+1}}|$. Thus, summing from $i = 1$ to d complete the conclusion. □

Proposition 2 *Consider a linear regression problem $\min_{\mathbf{w}\in\mathbb{R}^d} \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}}[(y - \mathbf{w}\mathbf{x})^2]$ with data pairs $\{(\mathbf{x}, y)\}$, where $\mathbf{x} \in \mathbb{R}^d$ is sampled from $\mathcal{N}(0, \sigma^2\mathbf{I})$ and $y \in \mathbb{R}$ is its response. Consider using LBS and DNAS to quantize the linear regression weights. Let $\mathbf{w}_L^t$ and $\mathbf{w}_D^t$ be the quantized regression weights of LBS and DNAS at the iteration t of the optimization, respectively. Then the following equivalence holds during the optimization process*

$$\begin{aligned}
&\lim_{t\to\infty} \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}}[(y - \mathbf{w}_L^t\mathbf{x})^2] = \lim_{t\to\infty} \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}}[(y - \mathbf{w}_D^t\mathbf{x})^2], \\
&\text{where } \mathbf{w}_L^t = \mathbf{w}_{b_1}^t + g_{b_2}^t(\mathbf{r}_{b_2}^t + \cdots + g_{b_{K-1}}^t(\mathbf{r}_{b_{K-1}}^t + g_{b_K}^t\mathbf{r}_{b_K}^t)), \\
&\qquad \mathbf{r}_{b_j}^t = D(\mathbf{w}_L^t - \mathbf{w}_{L,b_{j-1}}^t, s_{b_j}),\ j = 2, \cdots, K, \\
&\qquad \mathbf{w}_D^t = \sum_{i=1}^K p_i^t \mathbf{w}_{b_i}^t, \quad \sum_{i=1}^K p_i^t = 1.
\end{aligned} \tag{27}$$

Proof: We first rewrite the optimization objective as:

$$(y - \mathbf{w}^t\mathbf{x})^2 = (\mathbf{w}^*\mathbf{x} - \mathbf{w}^t\mathbf{x} + \Delta_\mathbf{x})^2 = \left((\mathbf{w}^* - \mathbf{w}^t)\mathbf{x}\right)^2 + 2\Delta_\mathbf{x}\left\langle \mathbf{w}^* - \mathbf{w}^t, \mathbf{x}\right\rangle + \Delta_\mathbf{x}^2 = (\mathbf{r}^t\mathbf{x})^2 + 2\Delta_\mathbf{x}\left\langle \mathbf{r}^t, \mathbf{x}\right\rangle + \Delta_\mathbf{x}^2,$$

where $\mathbf{w}^*$ is the optimal value of $\mathbf{w}$, $\Delta_{\mathbf{x}}$ is the regression error that is a constant, and $\mathbf{r}^t$ denotes the weight quantization error at the t-th iteration. Taking expectation, we have following results

$$\begin{aligned}
\mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}}[(y-\mathbf{w}^t\,\mathbf{x})^2] &= \mathbb{E}_{\mathbf{x}\sim\mathcal{N}(0,\sigma^2\mathbf{I})}(\mathbf{r}^t\,\mathbf{x})^2+2\Delta_{\mathbf{x}}\left\langle\mathbf{r}^t,\mathbf{x}\right\rangle+\Delta_{\mathbf{x}}^2 \\
&= \mathbb{E}_{\mathbf{x}\sim\mathcal{N}(0,\sigma^2\mathbf{I})}(\mathbf{r}^t\,\mathbf{x})^2+\Delta_{\mathbf{x}}^2 \\
&= \mathbb{E}_{\mathbf{x}\sim\mathcal{N}(0,\sigma^2\mathbf{I})}\left(\sum_{i=1}^{d}\mathbf{r}_i^t\mathbf{x}_i\right)^2+\Delta_{\mathbf{x}}^2 \\
&= \mathbb{E}_{\mathbf{x}\sim\mathcal{N}(0,\sigma^2\mathbf{I})}\sum_{i=1}^{d}(\mathbf{r}_i^t\mathbf{x}_i)^2+2\sum_{k\neq j}\mathbf{r}_k^t\mathbf{r}_j^t\mathbf{x}_k\mathbf{x}_j+\Delta_{\mathbf{x}}^2 \\
&= \sum_{i=1}^{d}(\mathbf{r}_i^t)^2\mathbb{E}_{\mathbf{x}\sim\mathcal{N}(0,\sigma^2\mathbf{I})}(\mathbf{x}^2)+2\sum_{k\neq j}\mathbf{r}_k^t\mathbf{r}_j^t\left(\mathbb{E}_{\mathbf{x}\sim\mathcal{N}(0,\sigma^2\mathbf{I})}(\mathbf{x})\right)^2+\mathbb{E}_{\mathbf{x}\sim\mathcal{N}(0,\sigma^2\mathbf{I})}\Delta_{\mathbf{x}}^2 \\
&= \sum_{i=1}^{d}(\mathbf{r}_i^t)^2\sigma^2+\mathbb{E}_{\mathbf{x}\sim\mathcal{N}(0,\sigma^2\mathbf{I})}\Delta_{\mathbf{x}}^2,
\end{aligned}$$

where we use assumption of Gaussian distribution $\mathbb{E}(\mathbf{x})=0$ and $\mathbb{E}(\mathbf{x}^2)=\sigma^2$.

The last equation suggests the training quantization error is only related to the weight quantization error $(\mathbf{r}_i^t)$ in an iteration since the term $\mathbb{E}_{\mathbf{x}\sim\mathcal{N}(0,\sigma^2\mathbf{I})}\Delta_{\mathbf{x}}^2$ is a constant. Followed by Lemma 2, the quantization error decreases as the bitwidth increases. This means that for each iteration, both the LBS and the DNAS are able to search for the K-bit to lower the quantization error by gradient descent.

Therefore, with gradient descent, for attribute $\eta/\sigma^2>0$, there exists some iteration T so that when $t>T$, we have

$$\left|\sum_{i=1}^{d}(\tilde{\mathbf{r}}_i^t)^2-\sum_{i=1}^{d}(\hat{\mathbf{r}}_i^t)^2\right|\leq\eta/\sigma^2,$$

where $\tilde{\mathbf{r}}=\mathbf{w}^*-\mathbf{w}_{\mathrm{L}}$ and $\hat{\mathbf{r}}=\mathbf{w}^*-\mathbf{w}_{\mathrm{D}}$ denote the weight quantization error with the LBS and the DNAS, respectively. Then multiplying σ^2 gives that

$$\left|\mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}}[(y-\mathbf{w}_L^t\,\mathbf{x})^2]-\mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}}[(y-\mathbf{w}_D^t\,\mathbf{x})^2]\right|\leq\eta.$$

Based on the definition of limit, the following result holds

$$\lim_{t\to\infty}\left|\mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}}[(y-\mathbf{w}_L^t\,\mathbf{x})^2]-\mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}}[(y-\mathbf{w}_D^t\,\mathbf{x})^2]\right|=0.$$

Therefore, we have

$$\lim_{t\to\infty}\mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}}[(y-\mathbf{w}_{\mathrm{L}}^t\,\mathbf{x})^2]=\lim_{t\to\infty}\mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}}[(y-\mathbf{w}_{\mathrm{D}}^t\,\mathbf{x})^2].$$

□

Algorithm 1 Training method for LBS

Input: A pre-trained model M^0, the number of candidate bitwidths K, a sequence of candidate bitwidths $\{b_j\}_{j=1}^K$, the number of epochs T, and hyperparameters λ.
Output: A compressed model M.

1: Initialize M using M^0.
2: Initialize threshold $\{\alpha_{b_j}^w=0\}_{j=1}^K$ for weights and threshold $\{\alpha_{b_j}^x=0\}_{j=2}^K$ for activations.
3: **for** $t=1,\ldots,T$ **do**
4: Update $\mathbf{W}$ and $\{\alpha_{b_j}^w\}_{j=1}^K$ by minimizing Eq. (16).
5: Update $\mathbf{W}$ and $\{\alpha_{b_j}^x\}_{j=2}^K$ by minimizing Eq. (16).
6: **end for**

7.4. More Details about the Differentiable Computational Cost

In this section, we introduce the details about the differentiable computational cost mentioned in Section 4.3. Following single-path NAS (Stamoulis et al., 2019), we model the computational cost as a function of binary gates. For simplicity, we consider performing weight quantization for layer l. The computational cost R^l for layer l is formulated as follows:

$$R^l = \sum_{c=1}^{G} g^l_{c,b_1}\big(R^l_{c,b_1} + g^l_{b_2}\big(R^l_{c,b_2} - R^l_{c,b_1} + \cdots + g^l_{b_K}(R^l_{c,b_K} - R^l_{c,b_{K-1}})\big)\big), \tag{28}$$

where R^l_{c,b_j} is the computational cost of the c-th group of filters with b_j-bit quantization, $g^l_{b_j}$ and $g^l_{c,b1}$ are binary gates for b_j-bit quantization and for pruning the c-th group of filters from layer l. As mentioned in Eq. 15, we approximate the gradient of the indicator function $\mathbb{1}(\cdot)$ using the gradient of the sigmoid function $S(\cdot)$. Therefore, the computational cost in the training objective remains differentiable.

7.5. More Details about the Training Algorithm for LBS

In this section, we introduce more details about the training algorithm for LBS. Let $\{\alpha^w_{b_j} = 0\}_{j=1}^K$ and $\{\alpha^x_{b_j} = 0\}_{j=2}^K$ be the thresholds for weights and activations, respectively. We use $\mathbf{W}$ to denote the model parameters. As mentioned in Section 4.3, the gradient approximation of the binary gate inevitably introduces noisy signals, which is even more severe when we quantize both weights and activations. Thus, we propose to train the binary gates of weights and activations in an alternative manner. The training algorithm of LBS is shown in Algorithm 1. An empirical study over the effect of alternative training scheme is put in Section 5.2.

7.6. More Experimental Details about LBS

Details about toy experiments As mentioned in Section 4.3, to study the difference between our single-path scheme and multi-path scheme in DNAS, we consider a linear regression scenario $\min_{\mathbf{w}\in\mathbb{R}^d} \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}}[(y - \mathbf{w}\mathbf{x})^2]$ and compare the quantization errors incurred by the two quantization schemes over the regression weights. First, we construct a toy linear regression dataset $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$. We randomly sample N=10,000 data $\mathbf{x} \in \mathbb{R}^{10}$ from Gaussian distribution $\mathcal{N}(0, \mathbf{I})$, and obtain $10,000$ corresponding responses $y = \mathbf{w}^*\mathbf{x} + \Delta$. Here, $\mathbf{w}^* \in \mathbb{R}^{10}$ is a fixed weight randomly sampled from $[0, 1]^{10}$ and $\Delta \in \mathbb{R}$ is a noise sampled from Gaussian distribution $\mathcal{N}(0, 1)$. Second, we apply LBS and DNAS to quantize the regression weights. The quantization errors of different methods are shown in Figure 2.

Details about experiments on CIFAR-100 and ImageNet. Following (Li et al., 2020; Esser et al., 2020), we introduce weight normalization during training. We use SGD with nesterov (Nesterov, 1983) for optimization, with a momentum of 0.9. For CIFAR-100, we use the same data augmentation as in (He et al., 2016), including translation and horizontal flipping. We first train the uncompressed network for 30 epochs on CIFAR-100 and 10 epochs on ImageNet. The learning rate is set to 0.001. We then fine-tune the searched compressed network to recover the performance decline. On CIFAR-100, we train the searched network for 200 epochs with a mini-batch size of 128. The learning rate is initialized to 0.1 and is divided by 10 at 80-th and 120-th epochs. For ResNet-18 on ImageNet, we finetune the searched network for 90 epochs with a mini-batch size of 256. For MobileNetV2 on ImageNet, we fine-tune for 150 epochs. For all models on ImageNet, the learning rate starts at 0.01 and decays with cosine annealing (Loshchilov & Hutter, 2017). Following (Liu et al., 2019a), we use zero initialization for the thresholds α.

7.7. More Details about Hardware-friendly Decomposition

In this section, we provide more explanations on hardware-friendly decomposition. As mentioned in Section 5, b_1 and γ_j can be set to arbitrary appropriate integer values (*e.g.*, 2, 3, etc.). By default, we set b_1 and γ_j to 2 for better hardware utilization. On general-purpose computing devices (*e.g.*, CPU, GPU), $byte$ (8 bits) is the lowest data type for operations. Other data types and ALU registers are all composed of multiple bytes in width. By setting b_1 and γ_j to 2, 2-bit/ 4-bit/ 8-bit quantization values can be packed into $byte$ (or $short, int, long$) data type without bit wasting. Otherwise, if b_1 and γ_j set to other values, it is inevitable to have wasted bits when packing mixed-precision quantized tensors on general-purpose devices. For example, one 32-bit int data type can be used to store ten 3-bit quantized values with 2 bits wasted. One might argue that these 2 bits can be leveraged with the next group of 3-bit data, but it will result in irregular memory access patterns, which will degrade the hardware utilization more seriously. Moreover, 8-bit quantization is able to achieve comparable performance with the full precision counterparts for many networks (Esser et al., 2020). Therefore, there is no need to

consider a bitwidth higher than 8.

7.8. More Details about Quantization Configurations

In this section, we provide more details about the quantization configurations of different methods. All the methods in Tables 1 and 2 use layer-wise and symmetric quantization schemes and the compared methods follow the quantization configurations in their papers. Specifically, for DQ (Uhlich et al., 2020), we parameterize the fixed-point quantizer using case U3 with $\boldsymbol{\theta} = [d, q_{\max}]$. We initialize the weights using a pre-trained model. The initial step size is set to $d = 2^{\lfloor \log_2(\max(|\mathbf{W}|)/(2^{b-1}-1)) \rfloor}$ for weights and 2^{-3} for activations. The remaining quantization parameters are set such that the initial bitwidth is 4-bit. For HAQ (Wang et al., 2019), we first truncate the weights and activations into the range of $[-v_w, v_w]$ and $[0, v_x]$, respectively. We then perform linear quantization for both weights and activations. To find more proper v_w and v_x, we minimize the KL-divergence between the original weight distribution $\mathbf{W}$ and the quantized one $Q(\mathbf{W})$. For DNAS (Wu et al., 2018), we follow DoReFa-Net (Zhou et al., 2016) to quantize weights and follow PACT (Choi et al., 2018) to quantize activations. We initialize the learnable upper bound to 1. For DJPQ (Ying et al., 2020) and Bayesian Bits (van Baalen et al., 2020), we directly get the results from original papers. For other methods in Tables 1 and 2, we use the quantization function introduced in Section 3. The trainable quantization intervals v_x and v_w are initialized to 1.

Table 6. Resource-constrained compression on BitFusion. We evaluate the proposed LBS under the latency- and energy-constrained and report the Top-1 and Top-5 accuracy on ImageNet.

Network	Method	Latency-constrained			Energy-constrained		
		Acc.-1 (%)	Acc.-5 (%)	Latency (ms)	Acc.-1 (%)	Acc.-5 (%)	Energy (mJ)
MobileNetV2	8-bit precision	71.5	90.1	24.9	71.5	90.1	37.0
	LBS (Ours)	**72.0**	**90.4**	**21.6**	**71.7**	**90.3**	**29.8**

7.9. Comparisons on Resource-constrained Compression

To demonstrate the effectiveness of our LBS on hardware devices, we further apply LBS to compress MobileNetV2 under the resource constraints on the BitFusion architecture (Sharma et al., 2018). Instead of using BOPs, we use the latency and energy on a simulator of the BitFusion to measure the computational cost. We report the results in Table 6. Compared with fixed-precision quantization, LBS achieves better performance with lower latency and energy. Specifically, LBS compressed MobileNetV2 with much lower latency and energy even outperforms 8-bit MobileNetV2 in the Top-1 and Top-5 accuracy.

Table 7. Comparisons of the search efficiency on CIFAR-100. The search cost are measured on a GPU device (NVIDIA TITAN Xp).

Network	Method	Search Cost (GPU hours)
ResNet-20	HAQ	5.8
	DQ	3.0
	DNAS	2.8
	LBS-Q (Ours)	0.8
	LBS-P (Ours)	0.2
	LBS (Ours)	1.0

7.10. Comparisons on Search Efficiency

To evaluate the efficiency of the proposed LBS, we compare the search cost of different methods and report the results in Table 7. From the results, the search cost of the proposed LBS is much smaller than the state-of-the-art methods. Moreover, compared with LBS-Q, LBS only introduces a small amount of computational overhead. These results show the superior efficiency of our proposed methods.

7.11. Comparisons on Memory Footprints

To further demonstrate the effectiveness of the proposed LBS, we can also use the total weights and activations memory footprints (Uhlich et al., 2020) to measure the computational cost. We apply different methods to compress ResNet-56 and report the results in Table 8. From the results, LBS compressed ResNet-56 outperforms other methods with fewer memory footprints. These results show the effectiveness of our proposed LBS in terms of memory footprints reduction.

Table 8. Comparisons of different methods w.r.t. memory footprints. We compress ResNet-56 using different methods and report the results on CIFAR-100.

Method	Memory footprints (KB)	M.f. comp. ratio	Top-1 Acc. (%)	Top-5 Acc. (%)
Full-precision	5653.4	1.0	71.7	92.2
4-bit precision	711.7	7.9	70.9±0.3	91.2±0.4
DNAS	708.9	8.0	71.5±0.2	91.3±0.1
HAQ	700.0	8.1	71.3±0.1	91.1±0.1
LBS-Q (Ours)	674.5	8.4	71.5±0.2	91.6±0.2
LBS (Ours)	**657.3**	**8.6**	**71.6±0.1**	**91.8±0.4**

Table 9. Comparisons of different methods in terms of MobileNetV3 on CIFAR-100.

Method	BOPs (M)	BOP comp. ratio	Top-1 Acc. (%)	Top-5 Acc. (%)
Full-precision	68170.1	1.0	76.1	93.9
6-bit precision	2412.6	28.3	76.1±0.0	93.7±0.0
DQ	2136.3	31.9	75.9±0.1	93.7±0.1
HAQ	2191.7	31.1	76.1±0.1	93.5±0.0
DNAS	2051.9	33.2	76.1±0.1	93.7±0.1
LBS-P (Ours)	59465.8	1.1	76.0±0.0	93.5±0.0
LBS-Q (Ours)	2021.9	33.7	76.1±0.1	93.7±0.1
LBS (Ours)	**2006.6**	**34.0**	**76.1±0.1**	**93.7±0.1**

7.12. More Results in terms of MobileNetV3 on CIFAR-100

To investigate the effectiveness of the proposed LBS on the lightweight models, we apply our methods to MobileNetV3 on CIFAR-100. Following LSQ+ (Bhalgat et al., 2020), we introduce a learnable offset to handle the negative activations in hard-swish. We show the results in Table 9. From the results, our proposed LBS outperforms the compared methods with much fewer computational cost, which demonstrates the effectiveness of the proposed LBS on the lightweight models.

7.13. More Details about the Compression Configurations of the Compressed Models

In this section, we illustrate the detailed compression configurations (*i.e.*, bitwidth and/or pruning rate) of each layer from the quantized and compressed ResNet-18 and MobileNetV2 in Figures 5, 6, 7 and 8. From the results, our LBS assigns more bitwidths to the weights in the downsampling convolutional layer in ResNet-18 and depthwise convolutional layer in MobileNetV2. Intuitively, one possible reason is that the number of parameters and computational cost of these layers are much smaller than other layers. Compressing these layers may lead to a significant performance decline. Moreover, our LBS inclines to prune more filters in the shallower layers of ResNet-18, which can significantly reduce the number of parameters and computational overhead. To further explore the relationship between pruning and quantization, we illustrate the detailed configurations of each layer from a more compact MobileNetV2, as shown in Figure 9. From the figure, if a layer is set to a high pruning rate, our LBS tends to select a higher bitwidth to compensate for the performance decline. In contrast, if a layer is set to a low pruning rate, our LBS tends to select a lower bitwidth to reduce the model size and computational costs.

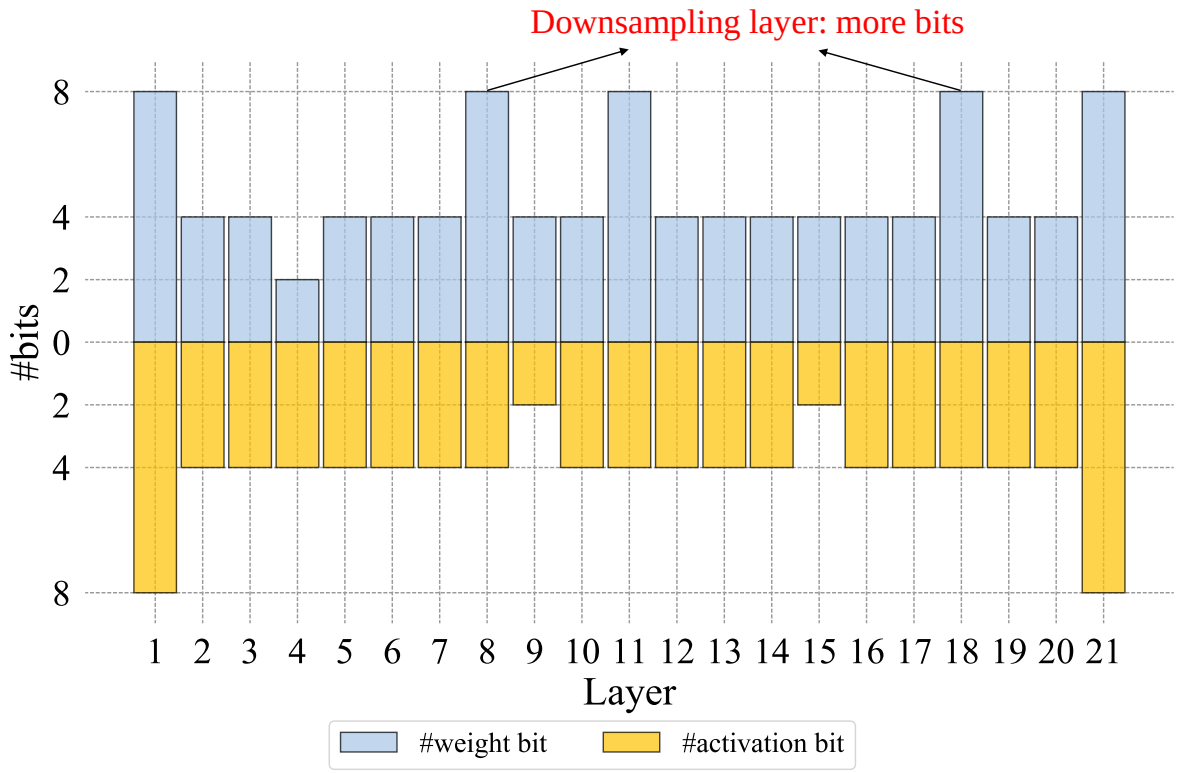


Figure 5. Detailed configurations of the quantized ResNet-18. The Top-1 accuracy, Top-5 accuracy and BOPs of the quantized ResNet-18 are 70.9%, 89.7% and 33.1G, respectively.

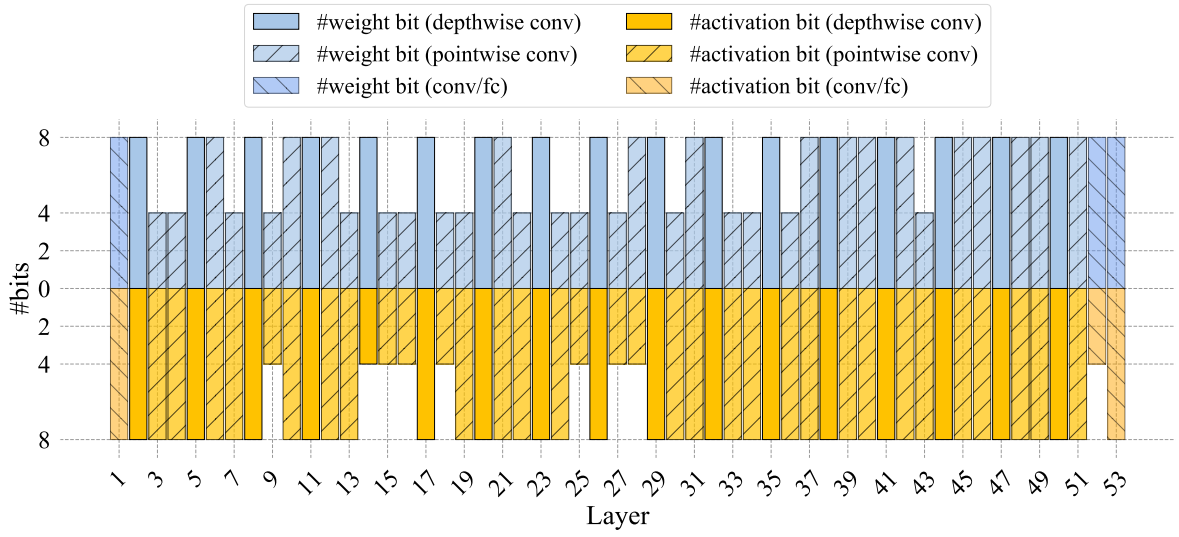
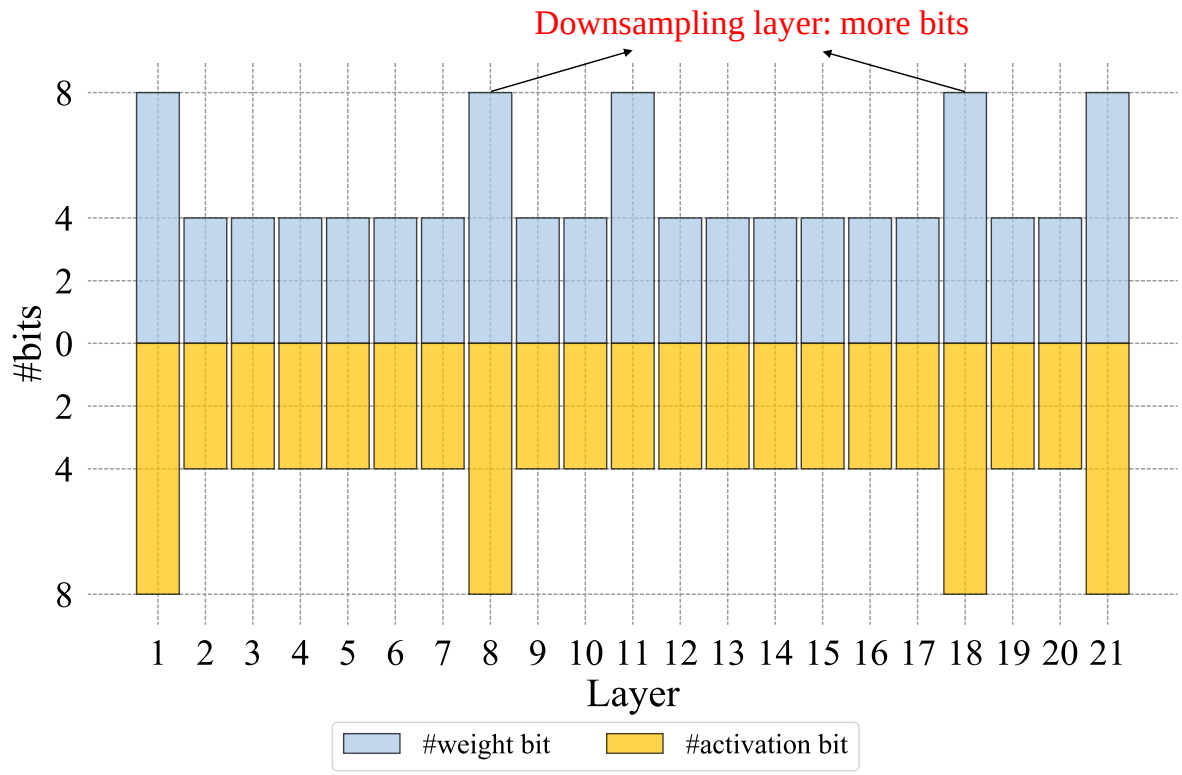
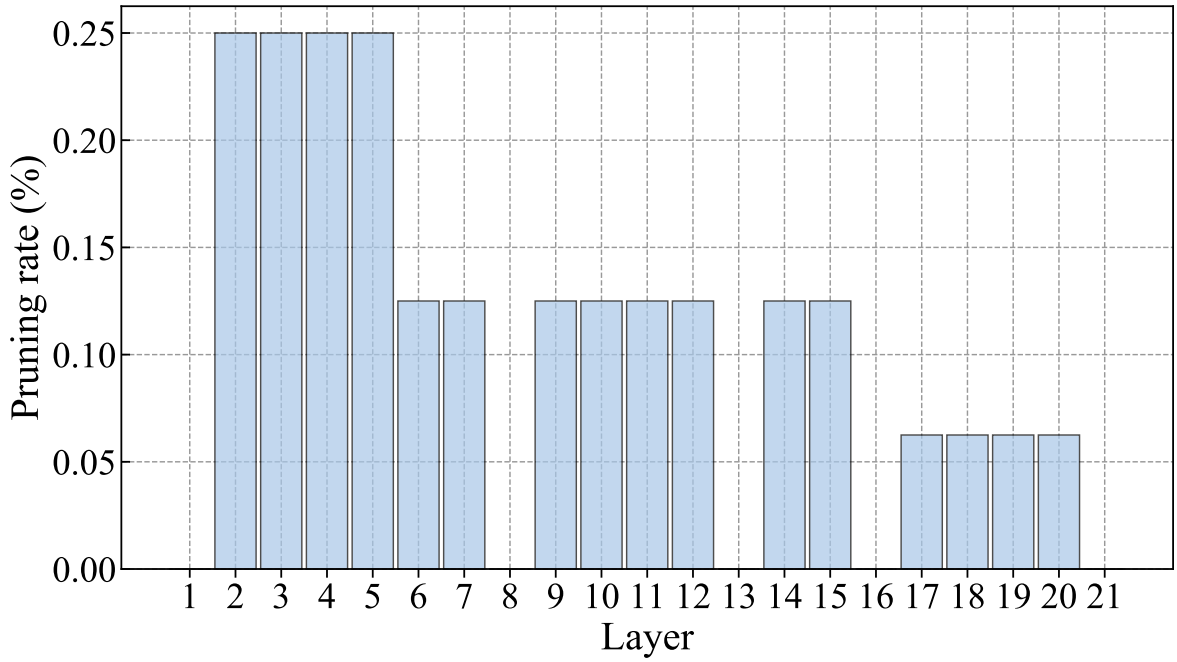


Figure 6. Detailed configurations of the quantized MobileNetV2. The Top-1 accuracy, Top-5 accuracy and BOPs of the quantized MobileNetV2 are 71.6%, 90.2% and 13.6G, respectively.

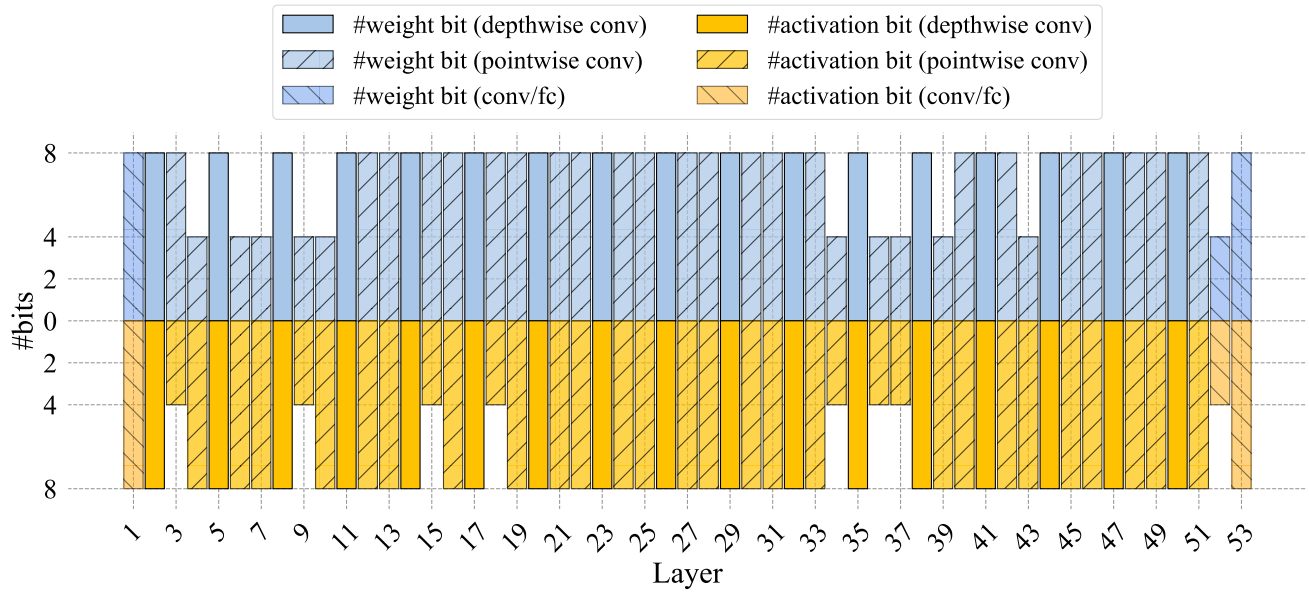


(a) Bitwidth configuration of the compressed ResNet-18.

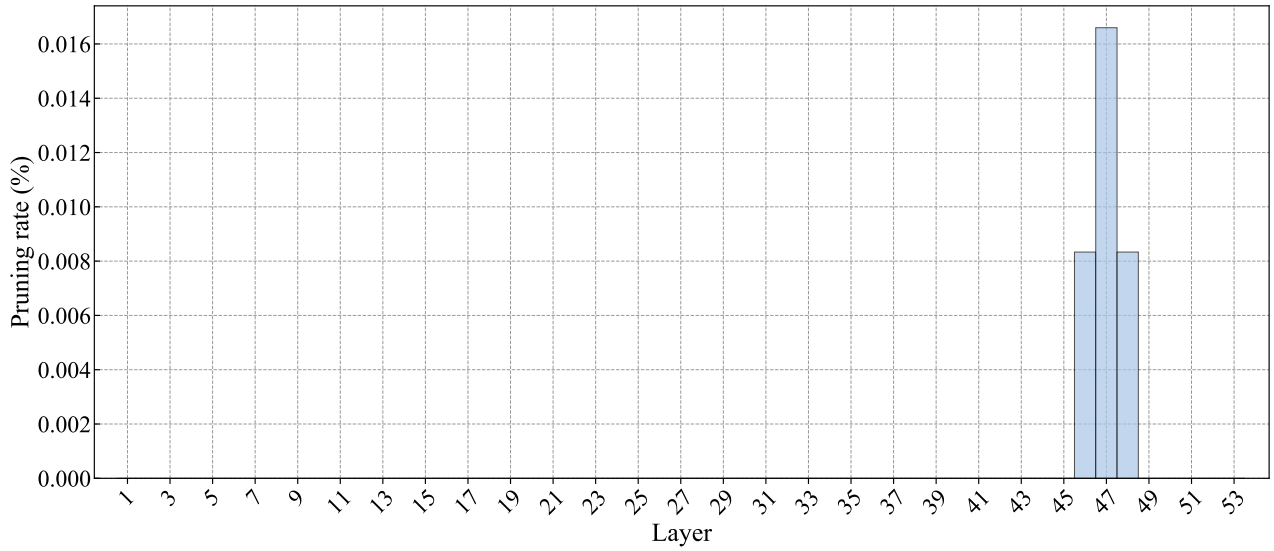


(b) Pruning rate configuration of the compressed ResNet-18. The pruning rate is defined as the ratio between #pruned weights of the compressed model and #weights of the uncompressed model.

Figure 7. Detailed configurations of the compressed ResNet-18. The Top-1 accuracy, Top-5 accuracy and BOPs of the compressed ResNet-18 are 70.8%, 89.6% and 32.3G, respectively.

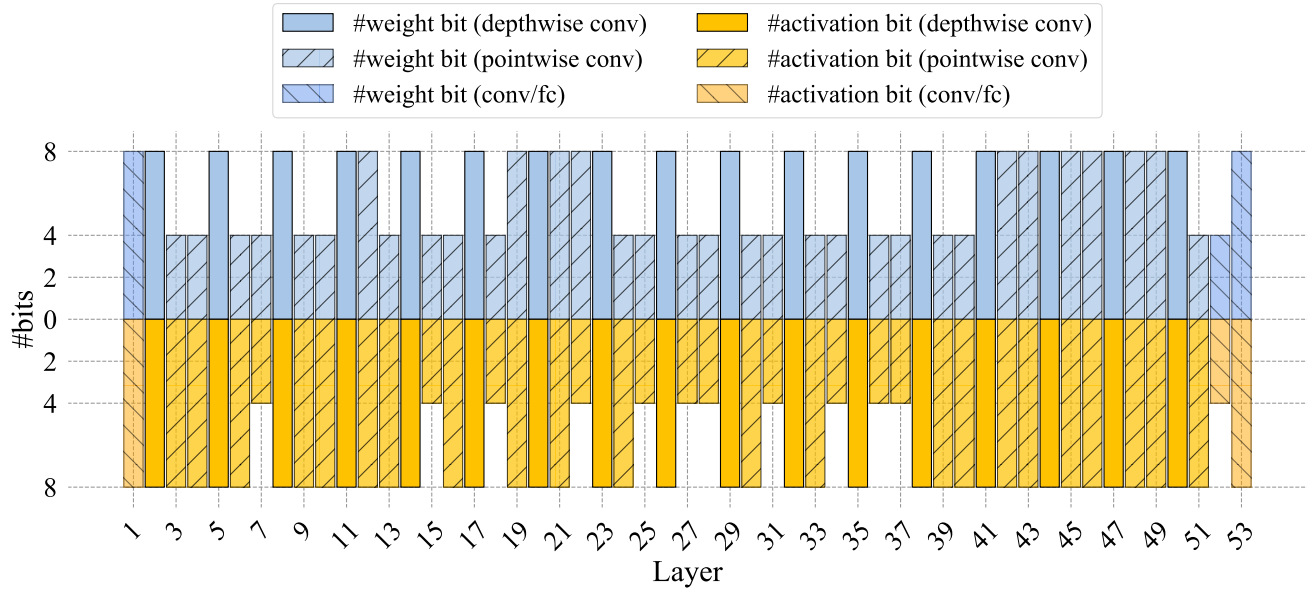


(a) Bitwidth configuration of the compressed MobileNetV2.

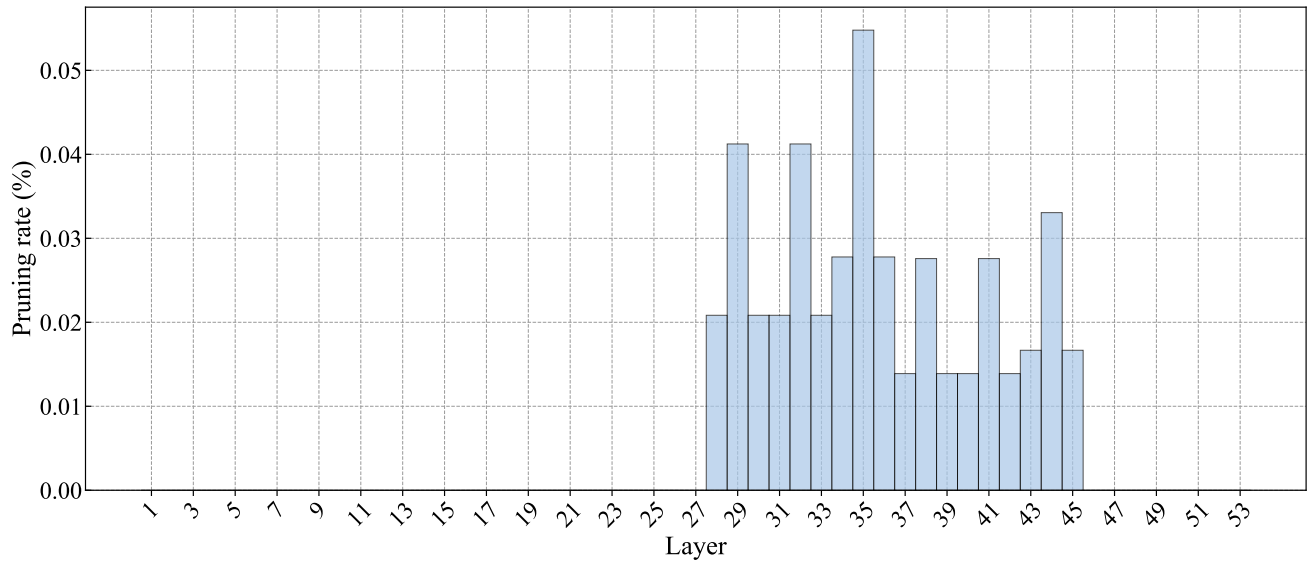


(b) Pruning rate configuration of the compressed MobileNetV2. The pruning rate is defined as the ratio between #pruned weights of the compressed model and #weights of the uncompressed model.

Figure 8. Detailed configurations of the compressed MobileNetV2. The Top-1 accuracy, Top-5 accuracy and BOPs of the compressed MobileNetV2 are 71.8%, 90.3% and 13.6G, respectively.



(a) Bitwidth configuration of the compressed MobileNetV2.



(b) Pruning rate configuration of the compressed MobileNetV2. The pruning rate is defined as the ratio between #pruned weights of the compressed model and #weights of the uncompressed model.

Figure 9. Detailed configurations of the compressed MobileNetV2. The Top-1 accuracy, Top-5 accuracy and BOPs of the compressed MobileNetV2 are 71.4%, 90.1% and 10.8G, respectively.