

# Efficient Bitwidth Search for Practical Mixed Precision Neural Network

Yuhang Li<sup>1,2\*</sup>, Wei Wang<sup>2</sup>, Haoli Bai<sup>3</sup>, Ruihao Gong<sup>1</sup>, Xin Dong<sup>4</sup>, and Fengwei Yu<sup>1</sup>

<sup>1</sup> SenseTime Research, Beijing, China

<sup>2</sup> National University of Singapore, Singapore

<sup>3</sup> The Chinese University of Hong Kong, Hong Kong, China

<sup>4</sup> Harvard University, Cambridge MA, USA

**Abstract.** Network quantization has rapidly become one of the most widely used methods to compress and accelerate deep neural networks. Recent efforts propose to quantize weights and activations from different layers with different precision to improve the overall performance. However, it is challenging to find the optimal bitwidth (i.e., precision) for weights and activations of each layer efficiently. Meanwhile, it is yet unclear how to perform convolution for weights and activations of different precision efficiently on generic hardware platforms. To resolve these two issues, in this paper, we first propose an Efficient Bitwidth Search (EBS) algorithm, which reuses the meta weights for different quantization bitwidth and thus the strength for each candidate precision can be optimized directly w.r.t the objective without superfluous copies, reducing both the memory and computational cost significantly. Second, we propose a binary decomposition algorithm that converts weights and activations of different precision into binary matrices to make the mixed precision convolution efficient and practical. Experiment results on CIFAR10 and ImageNet datasets demonstrate our mixed precision QNN outperforms the handcrafted uniform bitwidth counterparts and other mixed precision techniques.

**Keywords:** Efficient Inference, Quantized Neural Network, Differentiable Neural Architecture Search

## 1 Introduction

Due to the outstanding performance of deep neural networks (DNNs) in various applications, there is a surging demand for deploying DNN on edge/mobile devices. However, mobile devices typically have limited memory and energy, which requires DNNs to be compact and energy-efficient. There has been a large amount of research on compressing and accelerating neural networks, such as network pruning [8,34], matrix decomposition [16] and quantization [9,11]. Especially, network quantization is effective and feasible for deployment and has been widely studied in recent literature.

---

\* contact email loafyuhang@gmail.com.

Most existing quantization methods [13,15,32] adopt uniform precision quantization, i.e., a global precision<sup>5</sup> is used for weights and activations of all layers in CNNs. Recently, there is a trend of applying mixed precision quantization [28,5,30], i.e., assigning different bitwidths for the weights and activations across different layers. Mixed precision quantization is more flexible and has the potential to save more memory and computational cost without sacrificing the network’s expressiveness. There are two key problems (or challenges) to be resolved in mixed precision quantization as discussed below.

The first is *how to obtain the optimal bitwidth for each layer effectively and efficiently?* Existing techniques for mixed precision can be divided into rule-based methods and learning-based methods. Rule-based methods utilize a specific metric such as Hessian spectrum in [4] to determine the optimal bitwidth for each layer. However, these metrics rely on the heuristics provided by domain experts. Recently, inspired by Neural Architecture Search (NAS) [20,36,23,17,2], learning-based approaches [28,5,30] have been proposed for searching the bitwidths. Their methods are built upon either Deep Reinforcement Learning (DRL) or gradient based approaches. Despite their success, low efficiency and computational burden are the major drawbacks of these approaches. For instance, Hardware-aware Automated Quantization (HAQ) adopts DRL to search different configurations (i.e. quantization strategy), but each configuration retrains and evaluates a new model, which is time-consuming [28]. Recently proposed differentiable neural architecture search (DNAS) [30] follows the gradient-based approach to search for bitwidth configurations. However, the huge super net in DNAS still incurs heavy memory and computational cost for training [30]. To this end, we need an efficient way to determine the layerwise precision (i.e., bitwidth).

The second problem is *how to do convolution over weights and activations of mixed precision?* While [28] uses BISMO [27] and BitFusion [25] to support mixed precision computation, these platforms are specially designed. General platforms (like ARM CPU, GPU) do not welcome mixed precision computation as they only support INT8, INT4 instructions and binary operations in QNNs. Furthermore, they can only support weights and activations with the same precision. Considering that convolution is essential for CNN, it is necessary to have an efficient convolution implementation between  $M$ -bit quantized weights and  $K$ -bit quantized activations, where  $M$  and  $K$  are the optimized bitwidths.

In this paper, we propose two techniques to address the challenges mentioned above respectively. First, we propose Efficient Bitwidth Search, which is applied in the search process. To make the gradient-based bitwidth search algorithm [30] efficient, **we consider to jointly reduce the memory and computational cost.** On the one hand, similar to the spirit of weight sharing in [21,17], we maintain only one meta weights tensor that can adapt to branches of different quantization precision, which significantly reduces the memory from  $\mathcal{O}(N)$  to  $\mathcal{O}(1)$ , where  $N$  is the number of candidate bitwidths. On the other hand, instead of performing the convolution for each pair of weights precision and activations precision [30], we sum up the quantized weights (and activations) from all branches with Softmax

<sup>5</sup> In this work, we use precision and bitwidth in a mixed way.

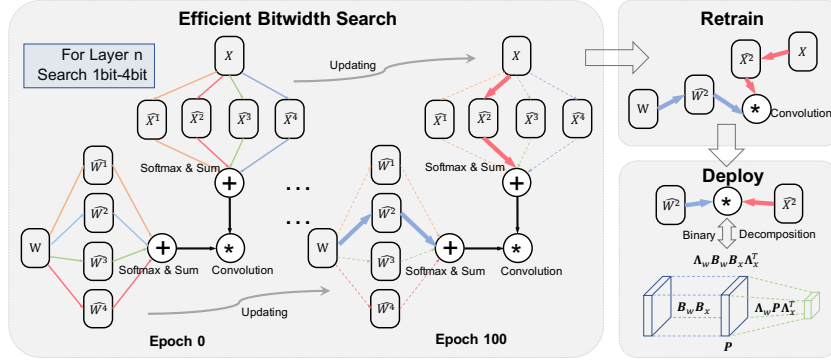


Fig. 1: The overall system pipeline. **In the search stage**, we only maintain one meta weight tensor and quantize it to different bitwidth, then we use softmax to relax the discrete selection of quantized weights. After epochs of training, the strength of each candidate precision is discriminative due to the gradient-based optimization. **In the retraining stage**, we only select the precision with largest strength and retrain the quantized network. **In the deployment stage**, Binary Decomposition is applied to support the mixed precision convolution.

weights, and then perform only one convolution, which theoretically reduces the computation from  $\mathcal{O}(N^2)$  to  $\mathcal{O}(1)$ . Moreover, we show that with the weighted sum of quantized weights (and activations), the expressiveness of the quantized network gets significantly improved. Therefore, EBS explores a wider range of quantized space during the search thanks to the dynamic and flexible quantization function.

Second, we propose a Binary Decomposition (BD) algorithm which provides a general computation pattern for mixed precision data in the deployment stage. BD converts the multi-bit weights and activation tensors into binary matrices; then the convolution can be conducted over the binary matrices efficiently. Consequently, we can do convolution over mixed precision weights and activations on general-purpose computing platforms without special hardware support. The overall system pipeline is illustrated in Fig.1, which consists of three stages, namely efficient bitwidth search, model retraining and deployment (using BD for convolution).

#### Contributions:

1. We propose an efficient gradient-based search algorithm to find optimal layer-wise bitwidth for mixed precision quantization. The algorithm reduces both the memory and computational cost significantly and is applicable to both weights and activations.
2. We propose a binary decomposition approach to support efficient convolution over mixed precision weights and activations on generic hardware.

3. Extensive experiments are conducted on CIFAR10 and ImageNet dataset. Our model has better accuracy-latency trade-off than uniform precision QNNs and other mixed precision QNNs.

## 2 Related Works

**Quantization** Compression via reducing the bitwidth of the parameters in the neural network could significantly reduce the memory cost and accelerate the inference. [15,22,33] restrict the weight to the binary (1 bit) or ternary (2-bit), however, they have a relatively large accuracy gap with the full precision model. [13,31] optimize the quantization parameters (levels, intervals) through gradient descent and bridge the gap with full precision models. Yet these methods all use the hand-crafted uniform bitwidth for all layers in the networks. Intuitively, different layers show various sensitivity to quantization, and mixed quantization is recently proposed to assign different layer-wise precision accordingly. To search for the layerwise precision, HAWQ [4] employs a rule-based method based on the Hessian spectrum of each layer to select the optimal bitwidth, which usually relies on domain expertise. More recent progress arises from approaches in neural architecture search.

**Neural Architecture Search** Recently, Zoph *et al.* [35] use Deep Reinforcement Learning (DRL) for NAS, which inspires a series of works for searching operators and connections of the architecture [21,26]. Similar DRL techniques are applied in network quantization as well. Hardware-aware Automated Quantization (HAQ) [29] adopts DRL for learning layer-wise bitwidth allocation, yet for each bitwidth configuration a new quantized network is retrained, which is time-consuming. Gradient-based optimization is another promising branch for neural architecture search [17,2], where both weights and architectures jointly learned by gradient descent. DNAS [30] is the first to use such differentiable methods to search the quantization precision, and is the most related work to ours. Comparing to DNAS, our method adopts one meta weights for different quantization branches, and perform only one convolution after the weighted sum of quantized weights and activations with different bitwidth. As a result, our method improves the memory cost of DNAS from  $O(N)$  to  $O(1)$ , and computational cost from  $O(N^2)$  to  $O(1)$  significantly, where  $N$  is the number of all candidate bitwidths.

## 3 Preliminaries

In this paper, we consider 2D convolution, in which the weights (a.k.a. kernels) tensor of a convolution layer has four dimensions, namely, the input channel, output channel, height and width respectively, denoted as  $\mathbf{W} \in \mathbb{R}^{c_i \cdot c_o \cdot k \cdot k}$ . Similar to DoReFa [32], we denote the quantization function of weights as  $Q(\mathbf{W}) = \hat{\mathbf{W}}$  where  $\hat{\mathbf{W}}$  is computed by normalizing the full precision weights into  $[-1, 1]$ , and then rounding the results to the nearest quantization levels. Activations (denoted

by  $\mathbf{X}$ ) of a layer are rectified by ReLU and therefore are non-negative. During quantization, they are clipped to  $[0, \alpha]$ , normalized to  $[0, 1]$ , and then rounded to the nearest quantization levels. This process is formulated as follows:

$$\hat{\mathbf{W}} = 2 \times \text{quantize}_b\left(\frac{\tanh(\mathbf{W})}{2 \max(|\tanh(\mathbf{W})|)} + \frac{1}{2}\right) - 1, \quad (1a)$$

$$\hat{\mathbf{X}} = \alpha \times \text{quantize}_b(\text{clip}(\mathbf{X}, 0, \alpha)/\alpha), \quad (1b)$$

$$\text{quantize}_b(x) = \frac{1}{2^b - 1} \times \text{round}((2^b - 1) \times x), \quad (1c)$$

where  $\alpha$  is the learnable clipping parameter for activations and  $b$  is the bitwidth of each weight (or activation) number after quantization. Note that  $\text{quantize}_b(\cdot)$  includes the de-quantize process which means integers are scaled by  $1/(2^b - 1)$ . Eq. 1a-1c are element-wise operations, where  $\max(\cdot)$  in Eq. 1a returns the max absolute weight in  $\mathbf{W}$ .  $\text{round}(\cdot)$  maps the value to the nearest integer with round half up. We can see that the whole quantization scheme is parameterized by the bitwidth  $b$  and the clipping parameter  $\alpha$ . In this paper, we focus on optimizing  $b$ .

Note that quantization process returns unsigned fixed-point numbers, which make the inference easy to speedup via hardware accelerators. Assume  $\hat{\mathbf{w}}$  is a vector of  $M$ -bit fixed-point integers s.t.  $\hat{\mathbf{w}} = \sum_{m=0}^{M-1} c_m(\hat{\mathbf{w}})2^m$  and  $\hat{\mathbf{x}}$  is a vector of  $K$ -bit fixed-point integers s.t.  $\hat{\mathbf{x}} = \sum_{k=0}^{K-1} c_k(\hat{\mathbf{x}})2^k$ , where each element of  $c_m(\hat{\mathbf{w}})$  (or  $c_k(\hat{\mathbf{x}})$ ) is either 0 or 1. According to [32], the dot product of  $\hat{\mathbf{w}}$  and  $\hat{\mathbf{x}}$  is

$$\hat{\mathbf{w}} \cdot \hat{\mathbf{x}} = \sum_{m=0}^{M-1} \sum_{k=0}^{K-1} 2^{m+k} \text{bitcount}[\text{AND}(c_m(\hat{\mathbf{w}}), c_k(\hat{\mathbf{x}}))] \quad (2)$$

When training QNN, we maintain the original full precision weights, called the meta weight  $\mathbf{W}$ . The gradient of the meta weights are computed via Straight-Through Estimator [1], defined as follows:

$$\frac{\partial L}{\partial \mathbf{W}} = \frac{\partial L}{\partial \hat{\mathbf{W}}} \frac{\partial \hat{\mathbf{W}}}{\partial \mathbf{W}}, \text{ where STE is } \frac{\partial \text{quantize}_b(x)}{\partial x} = 1_{x \leq \alpha}. \quad (3)$$

STE returns 1 when  $x \leq \alpha$  otherwise the gradient will be rectified. With the gradient, we can apply SGD to update the meta weights.

## 4 Methodology

### 4.1 Efficient Bitwidth Search (EBS)

Let  $\mathbf{O}$  be the output of the convolution of weights and activations (i.e.  $\mathbf{O} = \mathbf{W} * \mathbf{X}$ ). We use  $\hat{\mathbf{W}}^b$  and  $\hat{\mathbf{X}}^b$  to represent  $b$ -bit quantized weights and activations. Our task is to find out the best bitwidth configuration for each layer. We discuss quantization of weights and activations respectively in the following paragraphs.

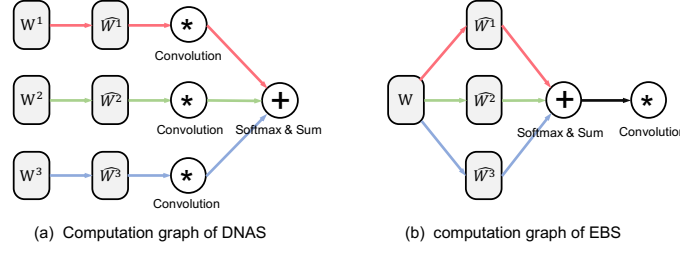


Fig. 2: Computation graphs. **(a)** DARTS and DNAS [17,30] maintain  $M$  full precision weight tensors and do  $M$  convolution operations if there are  $M$  candidate bitwidths; **(b)** Our method only stores one meta weight tensor; the quantized weights are aggregated; and thus only one convolution operation is done.

**Weights Quantization** A simple solution to optimize the bitwidth is to learn a strength (or importance) parameter for each bitwidth, and then select the bitwidth with the largest strength to quantize the matrix. Denote the bitwidth array as  $\mathcal{B} = [1, 2, \dots, b_{max}]$ , and the strength array as  $\mathbf{r}$ , the optimal bitwidth is

$$b^* = \mathcal{B}[\arg \max(\mathbf{r})]. \quad (4)$$

Nevertheless, since  $\arg \max(\cdot)$  is not differentiable, we cannot optimize the strength parameters using gradient-based methods. DARTS [17] and DNAS [30] resolve this issue by applying Softmax function over the learnable strength parameters  $\mathbf{r}$ , and then use them to scale the results from each operator correspondingly. In this way, we can compute the gradient of the loss w.r.t  $\mathbf{r}$  and apply SGD for optimization. Armed with the softmax trick, we can perform convolution as follows,

$$\mathbf{O} = \sum_{b_i \in \mathcal{B}} \mathbf{O}^i = \sum_{b_i \in \mathcal{B}} \frac{\exp(r_i)}{\sum_{b_j \in \mathcal{B}} \exp(r_j)} (\hat{\mathbf{W}}^i * \hat{\mathbf{X}}). \quad (5)$$

However, as shown in Fig. 2a, DNAS needs to store  $N$  copies of meta weights in the super net (one per branch), where  $N = |\mathcal{B}|$  is the number of branch candidates in DNAS. Moreover, if they further quantize activations, each output  $\mathbf{O}_{ij}$  is the feature map of  $\hat{\mathbf{W}}^i$  and  $\hat{\mathbf{X}}^j$ , there will be  $N^2$  convolution for a single convolutional layer. Therefore Eq. 5 suffers from  $\mathcal{O}(N)$  memory cost and  $\mathcal{O}(N^2)$  computation cost for each layer.

To prevent increasing GPU memory and computational cost, we only maintain one meta weight tensor as illustrated in Fig. 2b. In the forward pass, the quantized weight tensors of different precision are scaled and then summed before the convolution,

$$\mathbf{O} = \left( \sum_{b_i \in \mathcal{B}} \frac{\exp(r_i)}{\sum_{b_j \in \mathcal{B}} \exp(r_j)} \hat{\mathbf{W}}^i \right) * \hat{\mathbf{X}}. \quad (6)$$

Consequently, we reduce both the memory and computational cost to  $\mathcal{O}(1)$ , which significantly improves the search/training efficiency. Throughout the training process we keep the meta weight as full precision, where the back-propagated

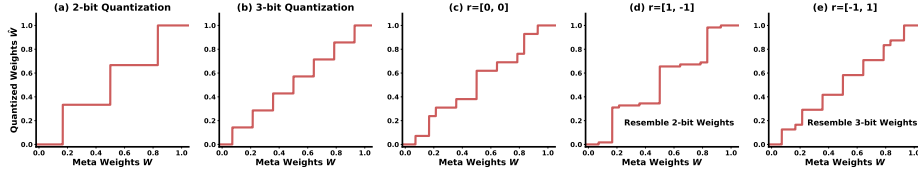


Fig. 3: Visualization of the aggregated quantization function.

gradients adjust itself to pick the most favorable quantization bitwidth. At the end of the training, we switch Softmax to max to select the best learned precision (i.e., branch) for the quantization network and remove the branches for other bitwidths. After that, we retrain the network weights to get the final mixed precision QNN. An example of the overall workflow is shown in Fig. 1. The strength for each candidate is distinctive (measured by the line-width), where 2-bit quantization for weights is more favorable by the training objective.

We need to highlight that, not only the design in Eq. 6 improves the efficiency, but it also brings a more flexible and dynamic quantization function, which shall benefit the search result. To see this, we visualize the aggregated quantization function of Eq. 6 for our EBS, as shown in Fig. 3. We find that the single precision quantization has the same effect as applying a step function with a uniform step size. Next, we use 2 bitwidths with  $\mathcal{B} = \{2, 3\}$ . We initialize the strength parameters to  $\mathbf{r} = [0, 0]$ . According to Eq. 6, the quantized weight is  $0.5\hat{\mathbf{W}}^2 + 0.5\hat{\mathbf{W}}^3$ , indicating the two quantization results are equally combined and therefore EBS has a larger capacity (i.e., finer precision) to explore the different bitwidths during training. As the training continues, the strengths get updated. When one strength parameter is much larger than the other one, e.g.  $\mathbf{r} = [-1, 1]$ , the aggregated quantization result is close to the quantization with the largest bitwidth. In summary, while EBS seeks to learn a single bitwidth to quantize the weights for inference, it explores multiple bitwidths during training, leading to a dynamic and flexible quantization function.

**Activation Quantization** We do quantization for the activations in the same way as weights quantization. A separate set of strength parameters are learned, denoted as  $\mathbf{s}$ . And only one convolution in EBS for each convolutional layer is computed during the search. The convolution of one layer is then formalized as

$$\mathbf{O} = \left( \sum_{b_i \in \mathcal{B}_w} \frac{\exp(r_i)}{\sum_{b_j \in \mathcal{B}_w} \exp(r_j)} \hat{\mathbf{W}}^i \right) * \left( \sum_{b_i \in \mathcal{B}_x} \frac{\exp(s_i)}{\sum_{b_j \in \mathcal{B}_x} \exp(s_j)} \hat{\mathbf{X}}^i \right). \quad (7)$$

**Stochastic Search** We also introduce a stochastic method to learn optimal precision. First, we denote  $f(\mathbf{r})$  as the softmax function. Distinguished from the deterministic search where  $\sum [f(\mathbf{r})] = 1$  is the coefficient for each candidate precision, we hereby model a categorical distribution where  $p_i = f(\mathbf{r})_i$  means the probability of being selected in the forward pass. Then Gumbel-Softmax

trick [18,12] is applied to estimate the gradient for sampling from a discrete distribution, given by

$$\hat{\mathbf{W}} = \sum_{b_i \in \mathcal{B}} \frac{(\exp(\log p_i + g_i)/\tau)}{\sum_{b_j \in \mathcal{B}} (\exp(\log p_j + g_j)/\tau)} \hat{\mathbf{W}}^i, \text{ where } g_i \sim \text{Gumbel}(0, 1). \quad (8)$$

Note that stochastic search is also applicable to activations.  $\tau$ , the temperature, controls the tightness in the sampling process. In experiments, we shall compare the differences of the two approaches.

## 4.2 Optimization

During training, we alternatively optimize weights and architecture parameters (i.e., the bitwidths), leading to a bilevel optimization problem. As aforementioned, bitwidth is closely related to the hardware performances of QNNs, such as model size and computational cost (FLOPs). Therefore, we add the computational cost into the loss function when optimizing the bitwidths. We first set a hyperparameter  $\text{FLOPs}_{\text{target}}$  for the desired computation cost in the target mixed precision QNN. The bilevel optimization is therefore given by:

$$\min_{\mathbf{r}, \mathbf{s}} \mathcal{L}_{\text{valid}}(\mathcal{W}^*, \mathbf{r}, \mathbf{s}) + \lambda \max(0, \text{FLOPs} - \text{FLOPs}_{\text{target}}), \quad (9)$$

$$\text{s.t. } \mathcal{W}^* = \arg \min_{\mathcal{W}} \mathcal{L}_{\text{train}}(\mathcal{W}, \mathbf{r}, \mathbf{s}), \quad (10)$$

where  $\mathcal{W}$  denotes the weights in the whole network. We abuse  $\mathbf{r}$  and  $\mathbf{s}$  to denote the strength parameters for all convolution layers. The detailed algorithm is shown in Alg. 1. We should point out that  $\text{FLOPs}_{\text{target}}$  cannot ensure the final QNN has exactly the same FLOPs as expected, because during inference  $\max(\mathbf{r})$  will result in a single precision weights while the FLOPs is computed based on the average as shown below. For FLOPs calculation, we use the expectation of the computational cost of all branches. We define a function  $\text{FLOP}(M, K)$  that returns the operation count of the convolution between  $M$ -bit weights and  $K$ -bit activations. From Eq. 2, we can see that the operation count is differentiable w.r.t  $M$  and  $K$ . We compute the expected FLOPs for both deterministic and stochastic search by

$$\mathbb{E}(\text{FLOPs}) = \text{FLOP}\left(\sum_{b_i \in \mathcal{B}_w} (f(\mathbf{r})_i \times b_i), \sum_{b_j \in \mathcal{B}_x} (f(\mathbf{s})_j \times b_j)\right). \quad (11)$$

## 4.3 Binary Decomposition

Our objective is to deploy the model for efficient and accurate inference. However, during inference, it is non-trivial to conduct efficient convolution between weights and activations of different precision on generic hardware. Note that the bitwidth for the weights and activations for the same layer could be different.



**Algorithm 1:** Search Algorithm.

---

**Input:** Initial strength  $\mathbf{r} = \mathbf{0}$ , target FLOPS; Total training epoch  $T$ ; training set and validation set.

**for** all  $i = 1, 2, \dots, T\text{-epoch}$  **do**

**for** all  $j = 1, 2, \dots, T'\text{-batch iteration}$  **do**

Update  $\mathcal{W}$  to minimize  $\mathcal{L}_{train}$  over the training set;

Update  $\mathbf{r}, \mathbf{s}$  to minimize Eq. 9 over the validation set;

**return** optimized bitwidths according to Eq. 4 for each convolution layer;

---

Img2col is a popular way to implement convolution that converts the weights tensor and activations tensor into matrices and then the convolution is done via matrix multiplication. We adopt img2col in this paper. For example, suppose we have a quantized weights matrix  $\hat{\mathbf{W}} \in S^{2 \times 3}$  and activations matrix  $\hat{\mathbf{X}} \in S^{3 \times 2}$ . We also assume that the bitwidths for the weights and activations are 2 and 3 respectively. So that  $S = \{0, 1, 2, 3\}$ . In Sec. 3 (Eq. 2, we show that the quantized values using fixed-point number format can be rewritten by  $\hat{\mathbf{W}} = \sum_{m=0}^1 c_m(\mathbf{W})2^m$  where  $c_m$  return the binary value  $\{0, 1\}$ . Hence, we use this expansion to decompose the weights and activations:

$$\hat{\mathbf{W}} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix} = \begin{bmatrix} 2^0 & 2^1 & 0 & 0 \\ 0 & 0 & 2^0 & 2^1 \end{bmatrix} \begin{bmatrix} c_0(w_{11}) & c_0(w_{12}) & c_0(w_{13}) \\ c_1(w_{11}) & c_1(w_{12}) & c_1(w_{13}) \\ c_0(w_{21}) & c_0(w_{22}) & c_0(w_{23}) \\ c_1(w_{21}) & c_1(w_{22}) & c_1(w_{23}) \end{bmatrix} = \mathbf{A}_w \mathbf{B}_w, \quad (12)$$

where  $\mathbf{B}_w \in \{0, 1\}^{4 \times 3}$  is the decomposed binary matrix and  $\mathbf{A}_w$  is the coefficient matrix for the binary values. The same decomposition can be applied to activations by  $\hat{\mathbf{X}} = \mathbf{B}_x \mathbf{A}_x^T$ . After BD, the feature maps is computed by

$$\mathbf{O} = \mathbf{A}_w \mathbf{P} \mathbf{A}_x^T \text{ where } \mathbf{P} = \mathbf{B}_w \mathbf{B}_x \in \mathbb{R}^{4 \times 4}, \quad (13)$$

$$o_{(i,j)} = \sum_{m=0}^1 \sum_{k=0}^1 p_{(2i+m-1, 2j+k-1)} 2^{m+k}. \quad (14)$$

We notice that the core computation is computed by binary operation. Next we introduce an efficient implementation of  $\mathbf{A}_w \mathbf{P} \mathbf{A}_x^T$ . Eq. 14 explicitly gives the outcome of  $\mathbf{A}_w \mathbf{P} \mathbf{A}_x^T$ . Here,  $i$  and  $j$  are the row and column index of matrix  $\mathbf{O}$  respectively. We visualize the computation for  $\mathbf{O}$  in the right figure. It can be seen that  $\mathbf{P}$  is divided into 4 parts, each of which does a vector dot product<sup>6</sup> with a  $2 \times 2$  matrix consisting of powers-of-2 values. This means we can use a **depthwise convolution** with stride 2 and powers-of-2 kernels to implement  $\mathbf{A}_w \mathbf{P} \mathbf{A}_x^T$ .

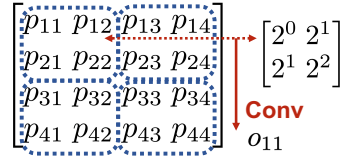


Fig. 4: Computation of Eq.14

<sup>6</sup> We flatten the matrices into vectors first.

We next present the formal definition of Binary Decomposition. BD uses 2 convolutions to obtain the final outcome of the mixed precision convolutions, the first is the normal convolution with binary weights and activations, then a depthwise convolution is applied to compute  $\mathcal{O}$ . Suppose weights and activations are quantized  $M$ -bit and  $K$ -bit, respectively, where  $\hat{\mathbf{W}} \in S^{c_o \times s}$  and  $\hat{\mathbf{X}} \in S^{s \times n}$ . After decomposition, the coefficient matrix can be represented by:

$$\mathbf{A}_w = \text{diag}(\overbrace{\delta_w, \dots, \delta_w}^{c_o}), \text{ where } \delta_w = [2^0, 2^1, \dots, 2^{M-1}]. \quad (15)$$

Therefore  $\mathbf{A}_w$  has shape of  $(c_o \times c_o M)$  and  $\mathbf{A}_x^T$  has shape of  $(nK \times n)$ . The binary matrices  $\mathbf{B}_w \in \{0, 1\}^{c_o M \times s}$  and  $\mathbf{B}_x \in \{0, 1\}^{s \times nK}$  can be derived by taking the expansion of fixed-point values. In the first convolution, only AND and bitcount are required to get the intermediate result  $\mathbf{P}$ . Then we construct a second layer, with only one depthwise kernel computed by  $\delta_w^T \delta_x$ . The kernel has a shape of  $(M, K)$  with stride of  $(M, K)$ . Kernels are comprised of powers-of-2 values, therefore the convolution is equal to shift the values in  $\mathbf{P}$  and can be efficient to implement. As a result, BD circumvents the direct mixed precision computation by decoupling the binary operations and the depthwise convolution.

**Complexities Analysis** In this section, we give the computation and memory complexities analysis w.r.t. the BD algorithm and we show that BD actually only incur negligible memory overhead. For a layer-wise clipping parameter  $\alpha$  in activations, we do not modify any memory cost nor the computation since it can be merged in Batch Normalization [10] and therefore we omit its analysis.

Let us take a look at memory first, the shape of the quantized weight matrix is  $(c_o \times s)$  after `img2col`, where  $s = c_i k^2$ . Before BD, the storage cost is  $sc_o M$  bits. After decomposition, weights are represented by  $\mathbf{A}_w$  and  $\mathbf{B}_w$ .  $\mathbf{B}_w$  has  $s \times c_o M$  binary (1 bit) weights; therefore it incurs the same memory cost as the weights before BD. The additional memory cost comes from  $\mathbf{A}_w$ ; during inference, we do not store the sparse matrix, instead according to Eq. 14, we only need to store  $MK$  power-of-2 values. Considering that  $M$  and  $K$  are small, e.g., smaller than 5 in our experiments, the additional memory cost is negligible. where in deployment it represents the kernel of the second convolutional layer. The kernel is determined by  $\delta_w^T \delta_x$  which only needs  $MK$  powers-of-2 values. As a result, the only additional memory requirement is the  $MK$  fixed-point numbers and can be negligible compared to weights (e.g., most layer in ResNet-34 has the weight shape of  $256 \times 256 \times 3 \times 3$ ).

For computation,  $\hat{\mathbf{X}}$  has the dimension of  $(s \times n)$  where  $n$  denotes the numbers of element in a single channel of feature maps. Then, the total FLOPs based on the Eq. 2 is  $snc_o MK$  AND operations,  $nc_o MK$  bitcount and shift-adds operations. When BD is applied to the QNN, the first convolution outputs  $\mathbf{P} \in \mathbb{R}^{c_o M \times nk}$ , therefore, we can compute the FLOPs is also  $snc_o MK$  AND operations and  $nc_o MK$  bitcount operations. No shift-adds in the first stage because weights and activations are binarized. In the second stage, each element in  $\mathbf{P}$

needs one time shift-add. Therefore, we can conclude that no extra computation cost is introduced in BD.

## 5 Experiments

In this section, we evaluate our proposed EBS for the modern popular neural network ResNets [7] on ImageNet ILSVRC-2012 dataset [24] and CIFAR10 [14] to demonstrate the effectiveness of our algorithm.

**Implementation** Based on prior works and our preliminary experiment results, 5-bit quantization can generally preserve the full precision accuracy of ResNets. Therefore our search space is set to  $\mathcal{B} = \{1, 2, 3, 4, 5\}$ . We provide the details of the evaluation and search implementation as well as code in the Supplemental materials.

### 5.1 CIFAR10

In this section, we compare the accuracy and FLOPs of inference in ResNet-20, 32, and 50. The compared methods include: 1) uniform precision QNNs, which have a pre-defined bitwidth for all weights and activations, 2) EBS-Det: EBS with deterministic search, 3) EBS-Sto: EBS with stochastic search and 4) random search: sample a random precision QNN within the target FLOPs range. We run the last 3 algorithms with 3 different FLOPs targets.

Table 1 and Fig. 5 summarize the results of these 4 approaches. First, we compare our EBS-Det with uniform precision QNN. It can be seen that 2-bit quantized models have severe accuracy degradation (2.04%), while EBS-Det with similar computation cost can achieve 0.75% accuracy improvement. This means that 2 bit quantization is not enough for some weights and activations to extract or preserve sufficient information, and a reasonable precision allocation is important. Furthermore, EBS-Det can reach the full precision model accuracy with 4x speedup, which demonstrates it is not necessary to allocate 5-bit quantization to all weights and activations. In ResNet-32 and 56, the same trend is also observed. In particular, EBS-Det with 6.79x speedup can surpass the accuracy of 4-bit quantized model, which is a significant improvement.

We also compare deterministic search with stochastic search, EBS-Sto slightly outperforms EBS-Det when FLOPs are low. For example, in ResNet-56, EBS-Sto overshadows the deterministic method with 0.19% accuracy improvement as well as 1 million fewer FLOPs, one possible reason is that in the low bit scenario, the optimization falls in local minimum more frequently as it is difficult to optimize, and the stochastic method can somehow jump out from the local minimum. However, for 4-bit or 5-bit, the loss landscape resembles that of the full precision training, therefore, the deterministic search might become more suitable. Nevertheless, the discrepancies between these two methods are not significant. Last but not least important, we compare our search algorithm with random search, which initializes the model with a Gaussian vector of  $\mathbf{r}, f$  and sample the bitwidths to construct QNNs. We only keep QNNs whose FLOPs are

Table 1: Accuracy and computational cost comparison over CIFAR10.

Methods	Precision	ResNet-20			ResNet-32			ResNet-56		
		Accuracy	FLOPs	Saving	Accuracy	FLOPs	Saving	Accuracy	FLOPs	Saving
Full Prec.	32-bit	92.96	40.81 M	1.0×	93.52	69.12 M	1.0×	94.46	125.7 M	1.0×
Uniform Precision	5 bits	93.04	17.8 M	2.29×	93.47	30.5 M	2.27×	94.31	54.5 M	2.30×
	4 bits	92.72	11.6 M	3.53×	93.26	19.4 M	3.56×	93.87	35.0 M	3.59×
	3 bits	92.44	6.71 M	6.08×	92.77	11.1 M	6.23×	93.54	19.9 M	6.31×
QNN	2 bits	90.92	3.23 M	12.6×	91.58	5.18 M	13.3×	91.93	9.09 M	13.8×
	1 bit	84.31	1.14 M	35.8×	86.68	1.63 M	42.4×	88.14	2.60 M	48.3×
EBS-Det	<i>flexible</i>	<b>92.94</b>	<b>10.2 M</b>	<b>4.01×</b>	<b>93.53</b>	21.3 M	3.24×	<b>94.27</b>	33.4 M	3.76×
	<i>flexible</i>	<b>92.74</b>	6.72 M	6.07×	92.91	10.9 M	6.34×	94.05	18.5 M	6.79×
	<i>flexible</i>	91.67	<b>3.01 M</b>	<b>13.6×</b>	91.74	<b>4.51 M</b>	<b>15.3×</b>	93.25	9.02 M	13.9×
EBS-Sto	<i>flexible</i>	92.79	11.8 M	3.46×	93.37	<b>18.5 M</b>	<b>4.21×</b>	94.19	<b>32.0 M</b>	<b>3.93×</b>
	<i>flexible</i>	92.66	<b>6.23 M</b>	<b>6.56×</b>	<b>93.14</b>	<b>10.3 M</b>	<b>6.71×</b>	<b>94.09</b>	<b>16.1 M</b>	<b>7.80×</b>
	<i>flexible</i>	<b>91.91</b>	3.39 M	12.0×	<b>92.44</b>	5.01 M	13.8×	<b>93.44</b>	<b>8.04 M</b>	<b>15.6×</b>
Random Search	<i>flexible</i>	92.50	10.4 M	3.92×	93.15	18.0 M	3.84×	93.49	32.0 M	3.93×
	<i>flexible</i>	92.14	6.25 M	6.52×	92.40	10.4 M	6.64×	92.93	16.5 M	7.62×
	<i>flexible</i>	90.31	3.34 M	12.2×	91.56	5.48 M	12.6×	91.58	9.60 M	13.1×

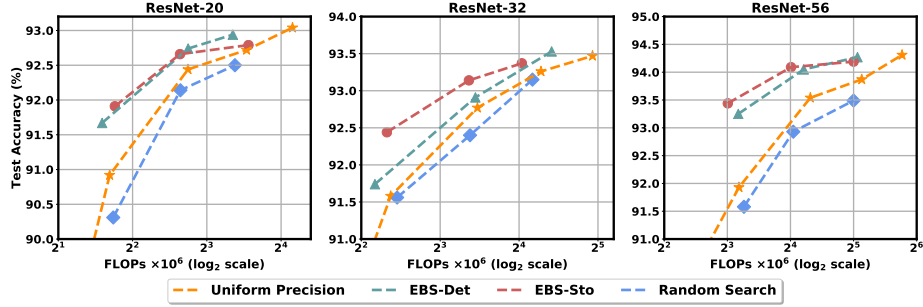


Fig. 5: Accuracy-FLOPs curve of ResNets on CIFAR10.

in target range. We can see that random searched QNNs have even lower accuracy than the uniform precision QNNs, it could be because some activations are sampled to 1-bit and thus severely damage the network performances as binary activations have the lowest expressiveness.

## 5.2 ImageNet

We evaluate our algorithm on ResNet-18 and ResNet-34 [7] for ImageNet dataset. Readers can refer to the Appendix for the results on ResNet-34. We compare our results with existing uniform precision QNN methods, including PACT [3] and other strong baselines such as LQ-Net [31] and DSQ [6]. We also compare our EBS algorithm with DNAS [30], another differentiable Mixed Precision QNN.

Table 5 summarizes the top-1, top-5 as well as the FLOPs of the baselines. Interestingly, PACT with 2-bit weights and 2-bit activations only achieve 64.4% top-1 accuracy on ImageNet, while its 1-bit weights and 3-bit activation version improves the accuracy by 0.9%. This may be counter-intuitive at the first glance, because the latter has fewer FLOPs and smaller model size than the

former one; however, [3,19] motioned that activations quantization is more important in QNNs, therefore increasing the bitwidth of activations can be helpful to QNNs. This phenomenon also points out that uniform precision for weights and activations is not optimal. Based on Fig. 6, we can see that EBS’s accuracy outperforms other uniform precision techniques, due to the reasonable precision allocation. In the low FLOPs model, EBS-Sto can surpass the state-of-the-art models by 1.8% for top-1 accuracy while preserving the FLOPs. Our EBS-Det can outstrip 5-bit PACT models with 0.4% for top-1 accuracy and only degrade 0.1% accuracy against the full precision models. Last, we compare our results with DNAS, with the label refinery enhancement for both approaches. It can be seen that our methods consistently improve the accuracy of DNAS, this is because our model can efficiently explore more combinations of weights and activations.

Methods	Precision		ResNet-18			
	Weights	Activations	Top-1	Top-5	FLOPs	Saving
Full Prec.	32-bit	32-bit	70.4	89.6	1.82 G	1.0×
PACT	5-bit	5-bit	69.8	89.3	849 M	2.14×
PACT	4-bit	4-bit	69.2	89.0	586 M	3.10×
LQ-Net	4-bit	4-bit	69.3	88.8	586 M	3.10×
DSQ	4-bit	4-bit	69.6	-	586 M	3.10×
EBS-Det	<i>flexible</i>	<i>flexible</i>	<b>70.2</b>	<b>89.3</b>	<b>558 M</b>	<b>3.26×</b>
EBS-Sto	<i>flexible</i>	<i>flexible</i>	70.0	<b>89.3</b>	564 M	3.22×
PACT	3-bit	3-bit	68.1	88.2	381 M	4.77×
LQ-Net	3-bit	3-bit	68.2	87.9	381 M	4.77×
DSQ	3-bit	3-bit	68.7	-	381 M	4.77×
EBS-Det	<i>flexible</i>	<i>flexible</i>	69.4	88.9	<b>369 M</b>	<b>4.93×</b>
EBS-Sto	<i>flexible</i>	<i>flexible</i>	<b>69.5</b>	<b>89.1</b>	380 M	4.78×
PACT	2-bit	2-bit	64.4	85.6	235 M	7.75×
PACT	1-bit	4-bit	65.0	85.9	235 M	7.75×
PACT	1-bit	3-bit	<b>65.3</b>	<b>85.9</b>	<b>206 M</b>	<b>8.83×</b>
LQ-Net	2-bit	2-bit	64.9	85.9	235 M	7.75×
DSQ	2-bit	2-bit	65.2	-	235 M	7.75×
EBS-Det	<i>flexible</i>	<i>flexible</i>	66.3	86.5	216 M	7.42×
EBS-Sto	<i>flexible</i>	<i>flexible</i>	<b>67.0</b>	<b>87.2</b>	<b>211 M</b>	<b>7.91×</b>
DNAS	<i>flexible</i>	<i>flexible</i>	70.6	-	594 M	3.06×
(+label refinery)	<i>flexible</i>	<i>flexible</i>	68.7	-	406 M	4.48×
EBS-Det	<i>flexible</i>	<i>flexible</i>	<b>71.1</b>	<b>89.7</b>	<b>558 M</b>	<b>3.26×</b>
(+label refinery)	<i>flexible</i>	<i>flexible</i>	<b>70.3</b>	<b>89.3</b>	<b>369 M</b>	<b>4.93×</b>

Table 2: Accuracy and computations cost comparison of ResNet-18 between EBS and existing methods in ImageNet.

The following figure gives the bitwidth distribution of ResNet-18 on ImageNet with the least FLOPs. We can see that most weights of the network are quantized to 1-bit and the average bitwidth of activations is higher than that of the weights. This allocation has also been confirmed by PACT that 1-bit weights and 4-bit activations are better than 2-bit weights and 2-bit activations. However, we can only know that activations need more bits, but how many bits for each layer and which layer should get more bits are challenging problems. EBS-Det and EBS-Sto can discover the precision distribution automatically and adapt to any target FLOPs.

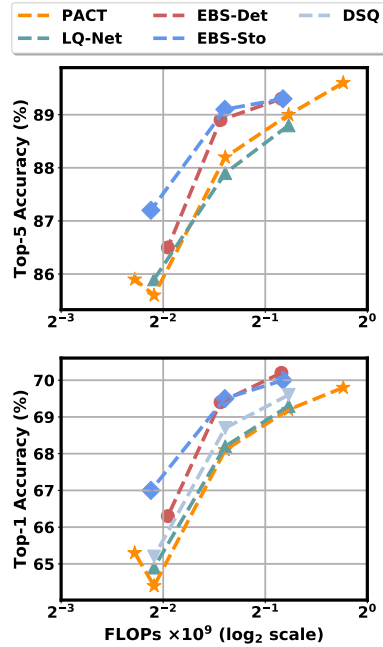


Fig. 6: Accuracy-FLOPs curve of ResNet-18.

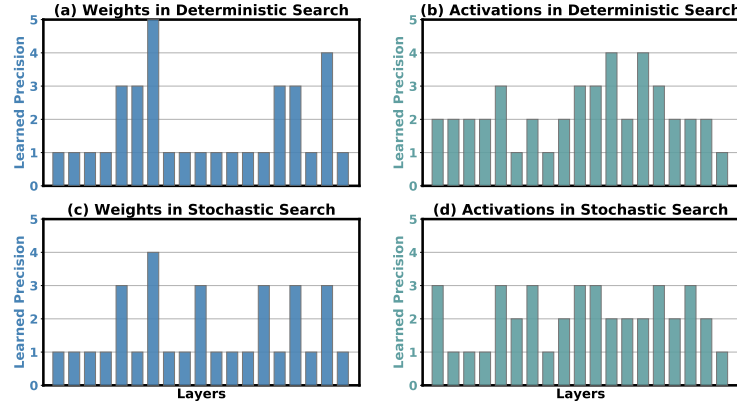


Fig. 7: Precision distribution of ResNet-18 with least FLOPs.

### 5.3 Efficiency

As we stated before, the GPU memory and computation cost are  $\mathcal{O}(N)$  and  $\mathcal{O}(N^2)$  in DNAS respectively; and both are reduced to  $\mathcal{O}(1)$  in EBS. It is necessary to verify the real GPU memory and time savings for DNAS and our EBS. Note that we use fake quantization training on GPU, where full precision with constrained values are used to emulate the quantization. We compare with the single-precision QNN, EBS and DNAS network as shown in Table 3. Here we let EBS and DNAS search in the same space (5 candidate precision per layer) and we show that EBS can reduce the orders of magnitude of GPU time and memory compared with DNAS. In particular, EBS only increase 3.6GB GPU memory and 1.4 seconds compared with uniform precision QNN when batch size is set to 32. If the batch size continues to 64 or 128, EBS can still search the bitwidth efficiently while DNAS model will become out of memory (OOM). Table 3: GPU memory (GB) and time (second) of training ResNet-18 for 10 iterations.

		Batch Size				Batch Size				
Model		16	32	64	128		16	32	64	128
Uniform Precision QNN	Memory	2.5	3.5	5.1	8.4	Time	16.7	20.9	26.7	42.0
	EBS	4.6	7.3	12.5	22.0	Time	17.7	22.3	30.7	47.1
DNAS	Memory	36.9	71.8	OOM	OOM	Time	55.5	100	-	-

## 6 Conclusion

In this paper, we propose a novel and efficient quantization scheme for mixed precision QNNs, which learns layer-wise bitwidths for weight and activation representation. We improve the efficiency of gradient-based search methods by reusing meta weights for different quantization bitwidth, which significantly reduces the memory and computation. To enable efficient convolution with mixed precision, we propose to decompose the tensors into a binary matrix and a coefficient matrix. Extensive experiments confirm the superiority of our solution in comparison to uniform precision and other mixed precision schemes.

## References

1. Bengio, Y., Léonard, N., Courville, A.: Estimating or propagating gradients through stochastic neurons for conditional computation. arXiv preprint arXiv:1308.3432 (2013)
2. Cai, H., Zhu, L., Han, S.: Proxylessnas: Direct neural architecture search on target task and hardware. arXiv preprint arXiv:1812.00332 (2018)
3. Choi, J., Wang, Z., Venkataramani, S., Chuang, P.I.J., Srinivasan, V., Gopalakrishnan, K.: Pact: Parameterized clipping activation for quantized neural networks. arXiv preprint arXiv:1805.06085 (2018)
4. Dong, Z., Yao, Z., Gholami, A., Mahoney, M., Keutzer, K.: Hawq: Hessian aware quantization of neural networks with mixed-precision. arXiv preprint arXiv:1905.03696 (2019)
5. Elthakeb, A.T., Pilligundla, P., Yazdanbakhsh, A., Kinzer, S., Esmaeilzadeh, H.: Releq: A reinforcement learning approach for deep quantization of neural networks. arXiv preprint arXiv:1811.01704 (2018)
6. Gong, R., Liu, X., Jiang, S., Li, T., Hu, P., Lin, J., Yu, F., Yan, J.: Differentiable soft quantization: Bridging full-precision and low-bit neural networks. arXiv preprint arXiv:1908.05033 (2019)
7. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
8. He, Y., Zhang, X., Sun, J.: Channel pruning for accelerating very deep neural networks. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 1389–1397 (2017)
9. Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., Bengio, Y.: Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research* **18**(1), 6869–6898 (2017)
10. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167 (2015)
11. Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., Kalenichenko, D.: Quantization and training of neural networks for efficient integer-arithmetic-only inference. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 2704–2713 (2018)
12. Jang, E., Gu, S., Poole, B.: Categorical reparameterization with gumbel-softmax. arXiv preprint arXiv:1611.01144 (2016)
13. Jung, S., Son, C., Lee, S., Son, J., Han, J.J., Kwak, Y., Hwang, S.J., Choi, C.: Learning to quantize deep networks by optimizing quantization intervals with task loss. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 4350–4359 (2019)
14. Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images. Tech. rep., Citeseer (2009)
15. Li, F., Zhang, B., Liu, B.: Ternary weight networks. arXiv preprint arXiv:1605.04711 (2016)
16. Liu, B., Wang, M., Foroosh, H., Tappen, M., Pensky, M.: Sparse convolutional neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 806–814 (2015)
17. Liu, H., Simonyan, K., Yang, Y.: Darts: Differentiable architecture search. arXiv preprint arXiv:1806.09055 (2018)

18. Maddison, C.J., Mnih, A., Teh, Y.W.: The concrete distribution: A continuous relaxation of discrete random variables. arXiv preprint arXiv:1611.00712 (2016)
19. Mishra, A., Nurvitadhi, E., Cook, J.J., Marr, D.: WRPN: Wide reduced-precision networks. In: International Conference on Learning Representations (2018), <https://openreview.net/forum?id=B1ZvaeAZ>
20. Pham, H., Guan, M.Y., Zoph, B., Le, Q.V., Dean, J.: Efficient neural architecture search via parameter sharing. arXiv preprint arXiv:1802.03268 (2018)
21. Pham, H., Guan, M.Y., Zoph, B., Le, Q.V., Dean, J.: Efficient neural architecture search via parameter sharing. arXiv preprint arXiv:1802.03268 (2018)
22. Rastegari, M., Ordonez, V., Redmon, J., Farhadi, A.: Xnor-net: Imagenet classification using binary convolutional neural networks. In: European Conference on Computer Vision. pp. 525–542. Springer (2016)
23. Real, E., Aggarwal, A., Huang, Y., Le, Q.V.: Regularized evolution for image classifier architecture search. In: Proceedings of the aaai conference on artificial intelligence. vol. 33, pp. 4780–4789 (2019)
24. 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. International journal of computer vision **115**(3), 211–252 (2015)
25. Sharma, H., Park, J., Suda, N., Lai, L., Chau, B., Chandra, V., Esmailzadeh, H.: Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network. In: 2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA). pp. 764–775. IEEE (2018)
26. Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., Le, Q.V.: Mnasnet: Platform-aware neural architecture search for mobile. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 2820–2828 (2019)
27. Umuroglu, Y., Rasnayake, L., Sjölander, M.: Bismo: A scalable bit-serial matrix multiplication overlay for reconfigurable computing. In: 2018 28th International Conference on Field Programmable Logic and Applications (FPL). pp. 307–3077. IEEE (2018)
28. Wang, K., Liu, Z., Lin, Y., Lin, J., Han, S.: Haq: Hardware-aware automated quantization with mixed precision. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2019)
29. Wang, K., Liu, Z., Lin, Y., Lin, J., Han, S.: Haq: Hardware-aware automated quantization with mixed precision. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 8612–8620 (2019)
30. Wu, B., Wang, Y., Zhang, P., Tian, Y., Vajda, P., Keutzer, K.: Mixed precision quantization of convnets via differentiable neural architecture search (2019), <https://openreview.net/forum?id=BJGVX3CqYm>
31. Zhang, D., Yang, J., Ye, D., Hua, G.: Lq-nets: Learned quantization for highly accurate and compact deep neural networks. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 365–382 (2018)
32. Zhou, S., Wu, Y., Ni, Z., Zhou, X., Wen, H., Zou, Y.: Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. arXiv preprint arXiv:1606.06160 (2016)
33. Zhu, C., Han, S., Mao, H., Dally, W.J.: Trained ternary quantization. arXiv preprint arXiv:1612.01064 (2016)
34. Zhuang, Z., Tan, M., Zhuang, B., Liu, J., Guo, Y., Wu, Q., Huang, J., Zhu, J.: Discrimination-aware channel pruning for deep neural networks. In: Advances in Neural Information Processing Systems. pp. 875–886 (2018)



- 35. Zoph, B., Le, Q.V.: Neural architecture search with reinforcement learning. arXiv preprint arXiv:1611.01578 (2016)
- 36. Zoph, B., Vasudevan, V., Shlens, J., Le, Q.V.: Learning transferable architectures for scalable image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 8697–8710 (2018)

## Appendix

### A Real World Latency

In this section, we report the latency when our mixed precision QNNs are deployed to edge devices. We test the inference speed on Raspberry Pi 3B with a 1.2 GHz 64-bit quad-core ARM Cortex-A53. We leverage the open sourced inference framework daBNN<sup>7</sup> and the SIMD instruction SSHL on ARM NEON to implement Binary Decomposition. We test the speed on several specific convolutional layers in ResNet-18, and we also report the inference speed of Bi-Real Net structure. It is necessary to point out there exists no open sourced library that supports the mixed precision computation in ARM CPU. We provide a general method and there is still a lot of room for optimization of the acceleration. Table 4 summarizes the latency tested on ARM CPU. We can see that the binary decomposition of *W1-A2* has approximately  $2\times$  latency of the binary convolution, which is similar to our theoretical estimation. Then, we test the overall inference speed of ResNet-18 under Bi-Real architecture. Note that the latency is not completely  $2\times$  because there are other overhead like img2col and data load/store.

Table 4: Latency tested in different types of layers in ResNet-18. *W1-A2* means weight are quantized to 1-bit and activations are quantized to 2-bit.

Layer Shape					Latency (ms)	
Kernel Size	Input Channels	Output Channels	Stride		<i>W1-A1</i>	<i>W1-A2</i>
3	64	64	1		5.76	11.65
3	128	128	1		5.43	11.46
3	256	256	1		5.73	11.76
3	256	512	2		1.65	3.45
3	512	512	1		7.10	14.35
Bi-Real-18-ImageNet					277.2	360.8

### B Implementation Details

#### B.1 Learning Clipping Parameter

Recent works successfully improve the task performance of QNN by learning the clipping parameter in activations like PACT. We also adopt this strategy in learning the clipping range of activations. In particular, only one clipping

<sup>7</sup> <https://github.com/JDAI-CV/dabnn>

parameter is maintained and learned. To specify the learning process, we split Eq. 1b to the following 3 sub-equations.

$$\tilde{\mathbf{X}} = \frac{\text{clip}(\mathbf{X}, 0, \alpha)}{\alpha}, \quad (16a)$$

$$\mathbf{X}^b = \text{quantize}_b(\tilde{\mathbf{X}}), \quad (16b)$$

$$\hat{\mathbf{X}} = \alpha \times \mathbf{X}^b. \quad (16c)$$

Here, we keep Eq. 16a and Eq. 16c intact in searching the bitwidth, i.e.,

$$\hat{\mathbf{X}} = \alpha \times \left( \sum_{b_i \in \mathcal{B}_x} \frac{\exp(s_i)}{\sum_{b_j \in \mathcal{B}_x} \exp(s_j)} \mathbf{X}^i \right), \text{ where } \mathbf{X}^i = \text{quantize}_i(\tilde{\mathbf{X}}). \quad (17)$$

Then, we use the Straight-Through Estimator to approximate the gradient of  $\alpha$ , given by

$$\frac{\partial \hat{\mathbf{X}}}{\partial \alpha} = \sum_{b_i \in \mathcal{B}_x} \frac{\exp(s_i)}{\sum_{b_j \in \mathcal{B}_x} \exp(s_j)} \left[ \mathbf{X}^i + \alpha \times \frac{\partial \mathbf{X}^i}{\partial \alpha} \right]. \quad (18)$$

Consequently, we have to consider two situations:  $\mathbf{X} > \alpha$  or  $\mathbf{X} \leq \alpha$ . In the former case, it is simple to compute that  $\tilde{\mathbf{X}} = \mathbf{X}^b = 1$  and thus the gradients is equal to 1. In the latter case, we can compute the gradient by:

$$\frac{\partial \hat{\mathbf{X}}}{\partial \alpha} = \sum_{b_i \in \mathcal{B}_x} \frac{\exp(s_i)}{\sum_{b_j \in \mathcal{B}_x} \exp(s_j)} \left[ \hat{\mathbf{X}}^i - \frac{\mathbf{X}}{\alpha} \right]. \quad (19)$$

## B.2 Implementation for Model Search

our search space is set to  $\mathcal{B} = \{1, 2, 3, 4, 5\}$ . The strength parameters (a.k.a architecture parameters)  $\mathbf{r}, \mathbf{s}$  are initialized to zero for both deterministic and stochastic optimization, as this would allocate each quantization bitwidth equal probability to be discovered. For CIFAR10 dataset that consists of 50K training images and 10K test images, we split the training images into half for training and half for validation respectively. We first pre-train a full precision model and use it to initialize the model for searching. We use SGD with momentum of 0.9 for weights and the learning rate is set to 0.01 followed by a cosine annealing schedule. For  $\mathbf{r}$ , we use Adam optimizer with learning rate of 0.02 and the tradeoff parameter  $\lambda$  is set to 0.06. The batch size is set to 64 for both training and validation. Weight decay for weights and  $\alpha$  is set to 5e-4. Target FLOPs is determined according to the FLOPs of 2-bit, 3-bit, and 4-bit architectures. We train the network for 60 epochs. The temperature parameter in Eq. 4.1 is linearly decreased to 0.4 from 1.0 in the stochastic search. Precision search only takes about 6 hours for ResNet-56 on a single NVIDIA GTX 1080Ti GPU.

For ImageNet dataset which contains of 1.2M training and 50K validation images, we follow a standard data preprocessing way used in baselines [7,13,22,31]. Training images are randomly cropped and resized and the validation images are center-cropped to 224×224. We also follow [30] where only 40 categories are

random sampled for searching. Note that we split 0.8x images to training set and 0.2x images to validation set, as we find that the validation loss cannot be minimized too much if we split a large set for validation. We set batch size to 256 for both training and validation. The initialization follows those in CIFAR10 experiments. We also use SGD with momentum of 0.9 to jointly train the weights and the clipping parameters, following by a cosine annealing schedule. Weight decay is set to  $1e-4$ . For  $\mathbf{r}$ , we use Adam optimizer with learning rate of 0.02 and the tradeoff parameter  $\lambda$  is set to 0.03. We do not quantize the first and the last layers as implemented in the prior works [3,30,32]. We search the model for 60 epochs. Precision search only takes about 10 hours for ResNet-18 on 4 Tesla V100 GPUs.

### B.3 Implementations for Model Retraining

During the model search, we save the strength parameters with the highest validation accuracy and directly use the arg max to select the optimal precision. For CIFAR10, we use the original 50K training images as the training set and 10K test images for the test. During model evaluation, there is no architecture parameter, and we do not have FLOPs penalty in the training objective. We use SGD with momentum of 0.9 to optimize the weights and the clipping parameters. The learning rate is set to 0.04 and followed by a cosine annealing learning rate. Weight decay (i.e., the L2 regularization term) is set to  $5e-4$  for high-bit models and  $1e-4$  for low-bit models as the low-bit QNN has less probability to overfit the training data. Batch size is set to 128 in the retraining. We use progressive initialization, that is to say, we progressively retrain the model from highest FLOPs (precision) and use it to initialize the next model. The first model are initialized from the full precision models. The clipping parameter  $\alpha$  is initialized to 6.0 in the first model. We train the model for 300 epochs. Regarding the uniform precision QNN and the random searched models in CIFAR10, we also use the same configuration in here.

For retraining the models on ImageNet dataset, we also followed the prior data preprocessing pipeline adopted in existing works. The test images are centered cropped to 224x224 and the training images are random resized and cropped to 224x224. Batch size is set to 1024, and the weight decay is set to  $1e-4$  to  $2e-5$  depends on the bitwidth. Other training configurations like learning rate and its scheduler are the same with CIFAR10 experiments. Note that we use label refinery in order to fairly compare the performance with DNAS.

## C Results of ResNet-34

We report the Top-1 accuracy and Top-5 accuracy on ResNet-34 in the following table. It can be seen that both EBS-Det and EBS-Sto consistently outperform the performance of other techniques, such as DNAS and DSQ with similar FLOPs.

Table 5: Accuracy and computations cost comparison of ResNet-34 between EBS and existing methods in ImageNet.

Methods	Precision		ResNet-34			
	Weights	Activations	Top-1	Top-5	FLOPs	Saving
Full Prec.	32-bit	32-bit	73.7	91.3	3.68 G	1.0×
BCGD	4-bit	4-bit	70.8	-	1096 M	3.36×
DSQ	4-bit	4-bit	72.8	-	1096 M	3.36×
EBS-Det	<i>flexible</i>	<i>flexible</i>	<b>73.5</b>	<b>91.2</b>	1104 M	3.33×
EBS-Sto	<i>flexible</i>	<i>flexible</i>	73.4	<b>91.2</b>	<b>1073 M</b>	<b>3.42×</b>
LQ-Net	3-bit	3-bit	71.9	90.2	669 M	5.50×
DSQ	3-bit	3-bit	72.5	-	669 M	5.50×
EBS-Det	<i>flexible</i>	<i>flexible</i>	73.0	<b>90.8</b>	654 M	<b>5.62×</b>
EBS-Sto	<i>flexible</i>	<i>flexible</i>	<b>73.1</b>	<b>90.8</b>	<b>648 M</b>	<b>5.67×</b>
LQ-Net	2-bit	2-bit	69.8	89.1	363 M	10.1×
LQ-Net	1-bit	2-bit	66.6	86.9	241 M	15.3×
DSQ	2-bit	2-bit	70.0	-	363 M	10.1×
EBS-Det	<i>flexible</i>	<i>flexible</i>	70.3	89.3	354 M	10.4×
EBS-Sto	<i>flexible</i>	<i>flexible</i>	<b>70.6</b>	<b>89.5</b>	<b>343 M</b>	<b>10.7×</b>
DNAS	<i>flexible</i>	<i>flexible</i>	74.1	-	1176 M	3.13×
(+label refinery)	<i>flexible</i>	<i>flexible</i>	73.2	-	825 M	4.46×
EBS-Det	<i>flexible</i>	<i>flexible</i>	<b>74.3</b>	<b>91.7</b>	1104 M	3.33×
(+label refinery)	<i>flexible</i>	<i>flexible</i>	<b>73.4</b>	<b>91.1</b>	654 M	5.62×