Quantization Through Search: A Novel Scheme to Quantize Convolutional Neural Networks in Finite Weight Space

Qing Lu
University of Notre Dame
Notre Dame, IN, USA
qlu2@nd.edu

Weiwen Jiang
George Mason University
Fairfax, Virginia USA
wjiang8@gmu.edu

Xiaowei Xu
Guangdong Provincial People's Hospital
Guangzhou
Guangdong, China
xiao.wei.xu@foxmail.com

Jingtong Hu
University of Pittsburgh
Pittsburgh, PA, USA
jthu@pitt.edu

Yiyu Shi
University of Notre Dame
Notre Dame, IN, USA
yshi4@nd.edu

ABSTRACT

Quantization has become an essential technique in compressing deep neural networks for deployment onto resource-constrained hardware. It is noticed that, the hardware efficiency of implementing quantized networks is highly coupled with the actual values to be quantized into, and therefore, with given bit widths, we can smartly choose a value space to further boost the hardware efficiency. For example, using weights of only integer powers of two, multiplication can be fulfilled by bit operations. Under such circumstances, however, existing quantization-aware training methods are either not suitable to apply or unable to unleash the expressiveness of very low bit-widths. For the best hardware efficiency, we revisit the quantization of convolutional neural networks and propose to address the training process from a weight-searching angle, as opposed to optimizing the quantizer functions as in existing works. Extensive experiments on CIFAR10 and ImageNet classification tasks are examined with implementations onto well-established CNN architectures, such as ResNet, VGG, and MobileNet, etc. It is shown the proposed method can achieve a lower accuracy loss than the state of arts, and/or improving implementation efficiency by using hardware-friendly weight values at the same time.

CCS CONCEPTS

• **Computing methodologies → Neural networks**; **Computer vision**; *Machine learning algorithms.*

KEYWORDS

Convolutional Neural Network, Quantization, Weight search

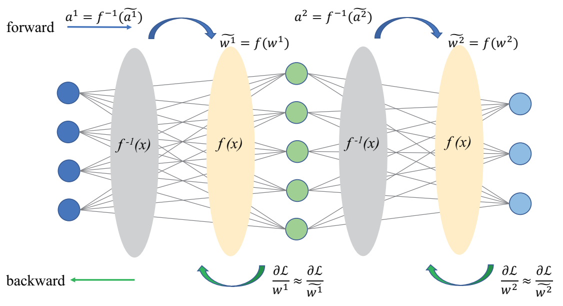
ACM Reference Format:
Qing Lu, Weiwen Jiang, Xiaowei Xu, Jingtong Hu, and Yiyu Shi. 2023. Quantization Through Search: A Novel Scheme to Quantize Convolutional Neural Networks in Finite Weight Space. In *28th Asia and South Pacific Design Automation Conference (ASPDAC '23), January 16–19, 2023, Tokyo, Japan.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3566097.3567870

1 INTRODUCTION

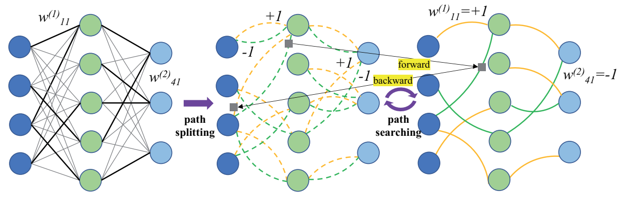
Recent years, Deep Neural Networks (DNNs) continuously refresh the-state-of-art performance in a variety of machine learning applications, such as image classification [9], object detection and segmentation [13], [15], speech recognition [3], etc. with the penalty of the ever-growing model sizes. This penalty has set prohibitive challenges to accommodate the model to resource-limited and power-hungry platforms like smart phones and Internet of Things (IoT) devices. As a response, model compression is essential and several compression techniques haven been proposed, e.g. pruning [16], knowledge distillation [6] and quantization [10], [14]. Among these techniques, quantization, a method that forces the data to take a set of discrete values, has been an active research topic due to its success in compressing the model size as well as simplifying computational complexity by large scales.

For example, the binary neural network (BNN) that constrains its weights and activations to be $\{-1, 1\}$ can roughly save 32× memory footprint and 1024× computation operations compared to the 32-bit full-precision counterpart. This is because the multiplication-and-accumulation (MAC) operations that dominate most state-of-art DNNs can be replaced by much more lightweight *xnor* and *popcount* operations. Such a computation and resource reduction is possible due the following three factors. (1) Binarizing the weights limits the data format for representation into 1 bit so the storage for weights is scaled down. (2) Quantizing the weights using $\{-1, 1\}$ omits the multiplication with the activations from the previous layer and turns MAC into accumulating/adding operations with proper sign flipping (*xnor*). (3) Reducing the bit width of activations linearly reduces the complexity of the addition operations such that to the extreme 1-bit accumulation boils down to logical operations (*popcount*). It is noted from these facts that binarizing weights into $\{-1, 1\}$ has the highest compression efficiency not only because each weight can be stored using 1-bit memory, but more importantly because the multipliers are the most area- and power- expensive units in hardware. Because of the architectural nature of convolutional neural networks (ConvNets), it is generally acknowledged that there are similar number of addition and multiplication operations in each inference.

According to the above analysis, given the quantization ratio, hardware execution efficiency can be maximized through constraining the weights into specific values that can be customized based on the target platforms. For example, using numbers with a magnitude of an integer power of two can implement multiplication with 1-bit logical operation and shift operation. Although such a requirement


forward
$a^1 = f^{-1}(\widetilde{a^1})$
$\widetilde{w^1} = f(w^1)$
$a^2 = f^{-1}(\widetilde{a^2})$
$\widetilde{w^2} = f(w^2)$
$f^{-1}(x)$
$f(x)$
$f^{-1}(x)$
$f(x)$
backward
$\frac{\partial \mathcal{L}}{w^1} \approx \frac{\partial \mathcal{L}}{\widetilde{w^1}}$
$\frac{\partial \mathcal{L}}{w^2} \approx \frac{\partial \mathcal{L}}{\widetilde{w^2}}$

(a) optimizing quantizer function



(b) searching weight values

Figure 1: Different design philosophies for quantizing neural networks: (a) optimizing the quantizer function vs (b) searching weight values.

can limit the expressiveness of a given neural network, recent study shows the approximation error can be an lower bounded as much as possible [4]. Therefore, quantization with awareness of weight value space is one of the most rewarding task for network compression. In this paper, we examine the process of quantizing the weights of DNNs from a novel viewpoint: instead of optimizing the quantizer functions that cannot guarantee the values weights are quantized into, we directly search values for them (Figure 1) Although solving the categorical search problem in deep neural networks has been attempted for a variety of applications such as neural architecture search, existing methods cannot be directly applied to weight search due to the huge search space. We claim that for gradient-based method, deterministic sampling is required and accordingly propose two specific algorithms namely *reparameterization with deterministic hard-sampling (RDHS)* and *stochastic gradient transfer (SGT)*. Our experiment on CIFAR10 and ImageNet shows that we can quantize the well-established ConvNets, including ResNet, VGG, and MobileNet with lower accuracy loss than state-of-art methods, and/or into more hardware-efficient value space.

2 QUANTIZATION THROUGH WEIGHT SEARCH

2.1 Problem Formulation

The weight quantization can be regarded as a search process in a discrete space $\{q^{(m)}\}^n$, where there are n parameters/weights each having m candidate values in the network. For each parameter, a value q from m optional candidate values need to be selected. Such a search space can be formulated as a path selection problem: for one pair of nodes/neurons, $\langle i, j\rangle$, we create m paths between them, with each path associated with a possible quantized value. The path selection problem is to determine a specific path for each pair of nodes in the network. During inference, the weight value associated to the selected path will be used for calculation.

For example, Figure 1b demonstrates the formulation of network binarization. For the shown network, we split each edge into two paths, representing weight values of +1 and −1. Let $o_{i,j}$ be the output of node i, the information passing from node i to j is either $1 \times o_{i,j}$ or $-1 \times o_{i,j}$. This is equivalent to the multiplication between the quantized weight w and the node output, i.e. $\mathbf{Q}(w) \times o_{i,j}$, where $\mathbf{Q}(.)$ is the mapping function such as the *Sign* function. Based on the above transformation, we can train the quantized neural networks by methods of searching in a discrete-valued space. After the path selection procedure, we determine the path between each pair of neurons, and consolidate the quantized network according to searching result.

2.2 Differentiable Search of Categorical Distribution

Considering the huge search space to be explored, we address the weight search problem using a differentiable method. We associate each node-to-node path with a parameter: for the path from node i to j, $\alpha^k_{i,j}$ indicates the possibility that the path k will be selected. As such, the path selection process can be expressed by a function f, where

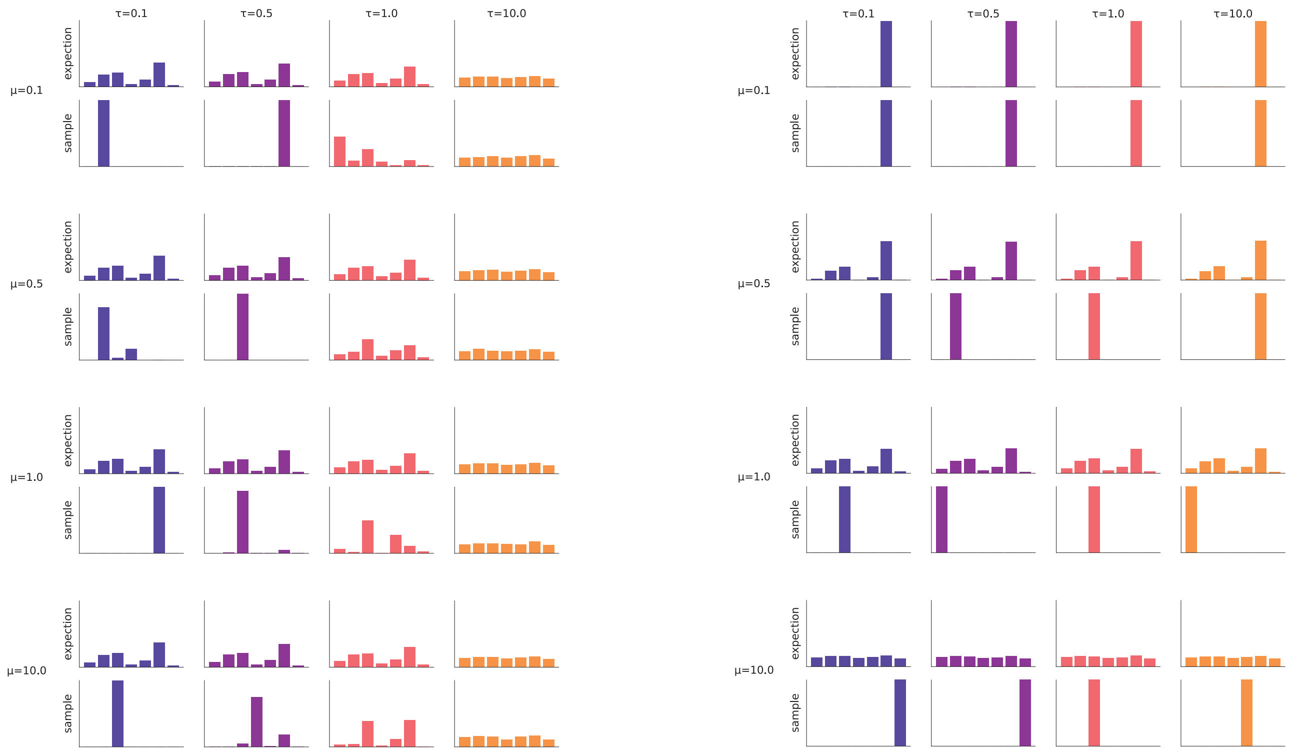
$$f(q^1, q^2, \cdots q^m | \alpha^1_{i,j}, \alpha^2_{i,j}, \cdots \alpha^m_{i,j}) = q^k. \tag{1}$$

The selection of k depends on the distribution of all $\{\alpha^{(m)}_{i,j}\}$ in a way that the higher $\alpha^k_{i,j}$, the more likely k is to be selected. With these parameters, the searching is equivalently optimizing them as regular training of a neural network using gradient decent. Due to the discrete the nature of this problem, f is not inherently differentiable, and we need to develop its gradient w.r.t. the associated parameters, i.e. $\frac{\partial \mathcal{L}}{\partial \boldsymbol{\alpha}_{i,j}}$, where $\boldsymbol{\alpha}_{i,j} = [\alpha^1_{i,j}, \alpha^2_{i,j}, \cdots, \alpha^m_{i,j}]^T$ and $\mathcal{L}$ is the loss function.

2.3 Reparameterization with Deterministic Hard-Sampling

Reparameterization is a widely used trick to compute the gradients of indifferentiable nodes with categorical distribution in neural networks. By using reparameterization, the samples can be drawn in a stochastic manner while the gradient can still be propagated through the node. Specifically, Gumbel-Softmax estimator proposed by [8] can generate samples by following their probabilities with the same expectation and has achieved state-of-art performance in a variety of discrete search problems of deep learning tasks, e.g. neural architecture search.

We apply the reparameterization trick to search weight values for quantized neural networks. Different from NAS, quantization requires absolutely discrete values to be sampled, so hard-sampling is used to control the path selection. As a result, parameter gradients need to be estimated such as in Straight-Through Gumbel Estimators. We generalize the mechanism of using Gumbel-softmax reparameterization here, and represent "soft" sample of each path



(a) Gumbel-Softmax **(b) Gumbel-Max**

Figure 2: The distribution of quasi Gumbel sampling methods using (a) Gumbel-Softmax and (b) Gumbel-Max are different shows the difference between quantization and other categorical search tasks. Gumbel-Max samples always has a one-hot categorical distribution and a constant expectation w.r.t temperature (τ). With lower reparameterization randomness (μ=0.1), a Gumbel-Max random variable converges to a categorical random sample following the same logits. As μ increases, Gumbel-Max samples become uniform and may not converge by training.

as

$$y_{i,j}^k = \frac{exp((log(\alpha_{i,j}^k) + \mu n_{i,j}^k)/\tau)}{\sum_k exp((log(\alpha_{i,j}^k) + \mu n_{i,j}^k)/\tau)} \tag{2}$$

In Equation (2), $n_{i,j}^k$ is the noise sample and μ is used for weighing the randomness in sampling. When $\mu = 1$ and $n_{i,j}^k \sim Gumbel(0, 1)$, it corresponds to standard Gumbel-Softmax relaxation and sample distribution will follow $\alpha_{i,j}^k$ in expected value when referring to the maximum $y_{i,j}^k$. The path parameters can be optimized by iterative forward and backward pass using gradient decent. In the forward pass, the weighting vector of the quantization values is computed as

$$z_{i,j} = \text{one\_hot}\left(\arg\max_k \left(y_{i,j}^k\right)\right). \tag{3}$$

In the backward pass, the gradient of softmax relaxation is approximated to that of the one-hot vector $z_{i,j}$, i.e.

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_{i,j}} \approx \frac{\partial \mathcal{L}}{\partial z_{i,j}}. \tag{4}$$

Such a network is end-to-end trainable, and the procedure is practically equivalent to regular full-precision training.

Interestingly, we found that standard Gumbel-Softmax sampling results in as poor empirical performance in accuracy as almost random. As one of the essential difference to similar tasks, searching weight values involves an exponentially higher volume of parameters that are deeply coupled. As a result, stochastic sampling will cause more highly shifting statistics as the network goes deeper, which makes convergence very hard. In Figure 2, we compare the distribution of samples using Gumbel-Softmax and Gumbel-Max under varying degrees of sampling stochasticity. In standard Gumbel distribution, it is observed that the main distinction is that the expectation of hard-sampling will not be controlled by the temperature τ and always keeps the true distribution. This means that each weight still has a relatively high probability to shift values even at a very low temperature, which may impede the convergence. When the stochasticity degree is pushed very high, e.g. $\mu = 10.0$, the Gumbel-Max sampling will be totally random throughout the training process, so the network will not converge at all. On the other hand, however if μ decreases, the expectation of Gumbel-Max will transform to be identical to each sample, and the state of each weight tends to be stable. We have empirically compared the training performance with different randomness in sampling and it is concluded that deterministic sampling ($\mu = 0$) is so far the best setting (see Section 3.3).

2.4 Stochastic Gradient Transfer

One of the problems in using reparameterization for weight searching is that it demands more memory and computation resources than training a full-precision network. This is because the node-to-node path splitting expands the computational graph with the growth of weight value space. As a matter of fact, since we sample the weight value of each pair of nodes in a deterministic manner, only one path needs to be involved in the computational graph while the state of other paths can be discarded during each forward/backward cycle, which will save some of the computation redundancy.

Driven by this observation, we derive the gradient of $\boldsymbol{\alpha}_{i,j}$ based on the gradient of $q^k_{i,j}$ that is selected in the forward pass because

$$k = \arg\max_r \alpha^r_{i,j}. \tag{5}$$

In the backward pass, the gradient of $q^k_{i,j}$ is transferred to every other path by following a heuristic: when the gradient of $q^k_{i,j}$ is positive, which suggests it should decrease, the paths with smaller candidate values should gain a higher odd and hence their associated parameters should increase; oppositely, the smaller candidate values should lose some odds and their associated parameters should decrease and vice versa. to indicate the direction and leave magnitude to be inherited. Suppose the selected value is q^k with $\alpha^k_{i,j} = \max\{\alpha^r_{i,j}\}$, the gradients will be

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\alpha}_{i,j}} = \frac{\partial \mathcal{L}}{\partial q^k_{i,j}} \cdot sign(\boldsymbol{q}_{i,j} - q^k_{i,j}) \tag{6}$$

where $\boldsymbol{q} = [q^1, q^2, \ldots, q^m]^T$ is the value vector.

2.5 Implementation Optimization for ConvNets

While the implementation of the path splitting and selection discussed in the above sections is straightforward, the implementation, especially with high dimensionality, still requires optimization. We take the standard 2D convolutions as an example to reveal the problem. Consider a convolutional layer with input feature map of size $H \times W$ and kernel size $K \times K$. There are (assuming 1-pixel stride and "same" padding mode) K^2HW connections within this layer, where K^2 weights are shared by these connections and each of them needs to be split into m paths. If we simply follow the implementation of node-to-node discretization to split paths, it is found that the kernel will render a total of m^{K^2} paths each mapping a template of the filter with discrete weights. For instance, a binarized convolutional kernel of 3×3 parameters has 512 different templates. Therefore, the size of the search space will grow at an exponential rate w.r.t K^2 and a polynomial rate w.r.t m.

Figure 3 shows how we manipulate the 2D convolution such that only m times more parameters are required for the training. Instead of splitting the kernel-wise operation, we split the whole image into m paths and generate m masks. Another way of looking at this process is that we expand the number of channels with each channel scaled by a discrete value of the value space. After that, we assign a real-valued kernel to each of the masks to perform the convolution respectively. The added kernels are related as a group that operates on a cluster of m different masks. Finally, the convolution of every mask-kernel pair in the group are processed together to produce one channel of the output of the whole layer. With such an arrangement, the same convolution operations are performed with $m\times$ more parameters during training.

3 EXPERIMENT

3.1 Setup Details

Dataset. We apply two commonly used computer vision datasets, namely CIFAR-10 and ImageNet (ILSVRC12) in the experiment. For CIFAR-10, We use the full volume of 50,000 images and 10,000 images as the training and validation set, respectively. For ImageNet there are 1.2 million images in the training set and 50,000 images in the testing set. The augmentation pipeline is similar for both datasets: images are firstly zero-padded by 4 pixels on each edge and then randomly cropped into the original size, flipped horizontally, and normalized.

Backbone Models. While the proposed method works with any model topology, we select some of the most representative networks for demonstration in this experiment. For CIFAR-10, we use VGG-small and ResNet20 as the backbone models following the implementations in LQ-Net [17] and HWGQ [1], respectively. For ImageNet, we show results on ResNet18, a benchmark network that frequently appears in related works [10], [7], [12], [17]. The original definition of theses networks are mostly kept unchanged except the forward/backward mechanism in the layers of interest. Following a common tradition, the first and last layers are not quantized.

Training Approach. All the trainable parameters are altogether updated using the SGD algorithm with the learning rate starting at 0.1 and decaying by a factor of 10. We set mini-batches of 128 and 256 images for the training on CIFAR-10 and ImageNet respectively. For the RDHS method, we initialize a temperature of 1 and anneal it down to 0.0001 by a decay factor of 0.97 at every epoch.

Value Space. The values of the quantized weights are constrained to subsets of the following numbers ± 1, $\pm\frac{1}{2}$, $\pm\frac{1}{4}$, and $\pm\frac{1}{8}$. The proposed method complies with arbitrary value space but we leave the exploration of values to future work and here concentrate only on these values characterized by hardware efficiency.

Comparisons with State-of-the-Art. We compare the performance of the proposed method with existing methods for neural network quantization. Considering the difference in the nature of the methods, we consider both the accuracy and the hardware resource for inference. For the same bit width, the memory size should be very close among all the methods, and is thus excluded. As the main hardware difference lies in the implementation of multiplications, we report the requirement of it to indicate the trade-off between value constraints and computational complexity.

3.2 Performance

Performance on CIFAR-10. Table 1 shows the accuracy and the hardware metrics of various methods in comparison with ours. Because of the easiness of this dataset, we only perform 1-bit quantization. The results suggest that weight searchign achieves an outstanding performance on VGG-small, with the highest accuracy and lowest accuracy loss among all the compared methods. For

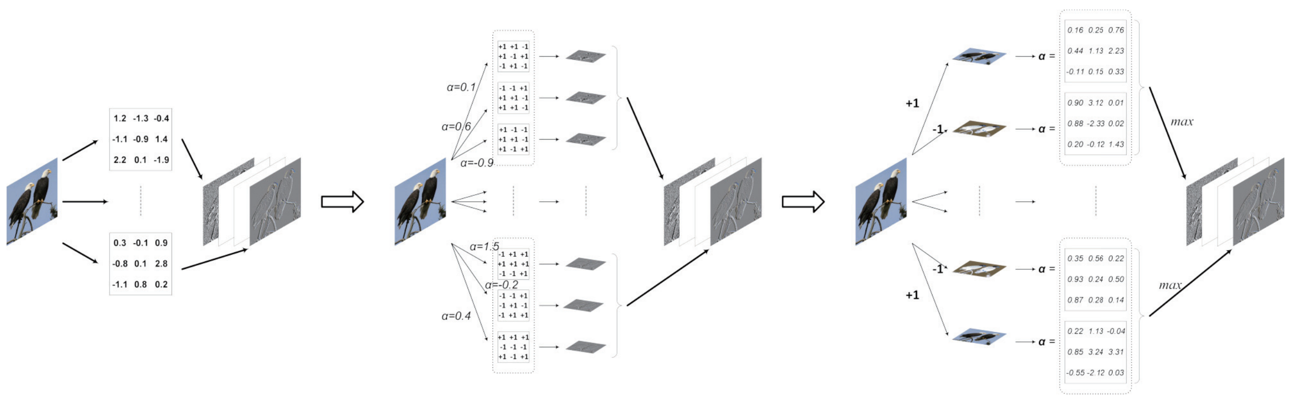


Figure 3: Path splitting of the 2D convolutional operation. Left: the flowchart of operations in a standard 2D convolution. Middle: the implementation of binarized convolution by splitting the kernels. Right: transforming kernel-based path splitting into convolution on channel-expanded input.

Table 1: Results using different methods on CIFAR-10: accuracy (%), accuracy loss (%) against unquantized network (FP32), and whether multiplication is required are reported.

Network	Method	Value Space	Acc. (%)	ΔAcc. (%)	Req. Mult
VGG-small	BC [2]	$\{\pm1\}$	91.7	2.1	N
	BWN [14]	$\mathbb{R}$	90.1	3.7	Y
	LAB [7]	$\{\pm1\}$	89.5	4.3	N
	LQ-Net [17]	$\mathbb{R}$	93.5	0.3	Y
	Ours	$\{\pm1\}$	**93.6**	**0.2**	N
ResNet-20	DoReFa [19]	$\mathbb{R}$	90.0	2.1	N
	LQ-Net [17]	$\mathbb{R}$	90.1	2.0	Y
	Ours	$\{\pm1\}$	**90.9**	**0.9**	N
MobileNet	*ours*	$\{\pm1\}$	93.78	0.3	N

Table 2: Results of ResNet18 quantized by different methods on ImageNet: Top-1/Top-5 accuracy (%) and accuracy loss (%) (shown in the parentheses) from the unquantized network.

#Bits	Method	Value Space	Top-1	Top-5	Req. Mult
1	BWN [14]	$\mathbb{R}$	60.8(8.5)	83.0(6.2)	Y
	ABC-Net [12]	$\mathbb{R}$	62.8(6.5)	84.4(4.8)	Y
	DSQ [5]	$\{\pm1\}$	63.7(6.2)	-	N
	Ours	$\{\pm1\}$	64.6(**4.5**)	85.4(**3.5**)	N
2	TWN [10]	$\mathbb{R}$	61.8(3.6)	84.2(2.6)	Y
	INQ [18]	$\{\pm1/2^n\}$	66.0(2.3)	87.1(1.6)	N
	TTQ [20]	$\mathbb{R}$	66.6(3.0)	87.2(2.0)	Y
	ADMM [11]	$\mathbb{R}$	67.0(2.1)	87.5(1.5)	Y
	ABC-Net [12]	$\mathbb{R}$	63.7(5.6)	85.2(4.0)	Y
	LQ-Net [17]	$\mathbb{R}$	68.0(2.3)	88.0(1.5)	Y
	Ours	$\{\pm1, \pm\frac{1}{2}\}$	67.4(**1.7**)	87.5(**1.4**)	N
3	INQ [18]	$\{\pm1/2^n\}$	68.1(0.2)	88.4(0.3)	N
	ABC-Net [12]	$\mathbb{R}$	66.2(3.1)	86.7(2.5)	Y
	LQ-Net [17]	$\mathbb{R}$	69.3(1.0)	88.8(0.7)	Y
	Ours	$\{\pm1, \pm\frac{1}{2}, \pm\frac{1}{4}, \pm\frac{1}{8}\}$	68.1(**1.0**)	88.2(**0.9**)	N

all the networks, our method has merely a 0.2%-0.3% degradation while most of the other methods suffer no less than 2%. On ther other hand, although LQ-Net has a comparable accuracy loss, its design constraint is less strict and thus requiring multiplication in hardware implementation.

Performance on ImageNet. The comparison of accuracy and hardware efficiency between different methods on ImageNet is shown in Table 2. For 1-bit quantization, our method has respective Top-1 and Top-5 accuracy loss of 4.5% and 3.5%, better than any of the other methods. Meanwhile, it saves computation compared with BWN and ABC-Net by eliminating the need of multiplications. Comparing with DSQ which uses the same value constraint, our method achieves 1.7% lower Top-1 accuracy loss. Similar observations can also apply to 2-bit cases where our method still yields the least Top-1 and Top-5 accuracy loss among all the methods. The only variation appears at the 3-bit cases where our method has similar and inferior accuraccy loss to INQ and LQ-Net. However, both of these two works have a larger value space for software design and correspondingly a lower efficiency for hardware design.

Further Analysis. Comparing Table 1 and Table 2, we can observe that, increasing the cardinality of the value space (i.e., increasing the number of bits) has a much stronger impact on accuracy than changing the values in the space. This is also expected as quantized neural networks, regardless of the weights used, always have universal approximability [4].

3.3 Ablation Study

We carry out additional experiments to investigate our methods in depth and explore further findings. There are two interesting topics. First, we study the trade-off between exploration and exploitation in sampling weight values. Second we compare the two proposed methods.

Table 3: Exploration of the relationship between accuracy and randomness in sampling. For all the test networks and bit-widths, the best accuracy occurs when deterministic sampling ($\mu = 0$) is adopted.

Network	#Bits	μ	Acc. (%)
VGG-small	1	0	93.6
		0.001	93.4
		0.01	85.6
ResNet20	1	0	90.9
		0.001	90.9
		0.01	90.7
		0.1	64.7
ResNet18	1	0	64.6/85.4
		0.001	64.1/84.1
		0.01	45.1/66.3
	2	0	67.4/87.5
		0.001	55.1/69.7
	3	0	68.1/88.2
		0.001	48.8/64.4

Exploration vs Exploitation To compare the impact randomness on accuracy performance, we have experimented using different sampling schemes. We show the training result of different networks with the same configuration but different levels of stochasticity in Table 3. All the networks show higher accuracy degradation when randomness increases and achieve the best result when $\mu = 0$. Therefore, we conclude that a deterministic sampling scheme is generally better than stochastic sampling scheme in weight searching for deep ConvNets.

RDHS vs SGT Lastly, we compare the two implementation algorithms proposed in this work with respect to their performance in accuracy, computation time, and memory footprints on GPU training. As demonstrated in Table 4, SGT has a clearly better efficiency in training time and memory usage than RDHS. Owing to its simplified computational graph, it saves 12% to 13% GPU hours to complete training and about 17% memory for all bit-widths. On the other hand, however, there is some degradation in accuracy. While both methods can achieve almost the same best accuracy, SGT training has a noticable lower average, and hence a higher variance than RDHS. As a result, we should use SGT over RDHS when there is a limited budget on resources for training but may need to repeat it for best resluts.

4 CONCLUSION

Quantizing the weights of a ConvNet into definite value space is more beneficial to hardware efficiency than other schemes but also harder to train. In this work, we solve this problem from the pure weight search angle. It is discovered that deterministic sampling is needed to for differentiable search method, and accordingly we have proposed two implementable algorithms. Experimental results on CIFAR10 and ImageNet shows our methods can quantize ConvNets with lower accuracy loss than state-or-art methods, and/or into more hardware-efficient value space.

Table 4: Full comparison between RDHS and SGT in training ResNet18 on ImageNet. Training time and memory footprint are reported in relative to training in full precision.

#Bits	Method	Avg. Acc. (%)	Best Acc. (%)	Time	Memory
1	RDHS	64.3/83.9	64.6/85.4	1.47	1.34
	SGT	64.1/81.8	64.5/85.4	1.33	1.10
2	RDHS	67.5/87.0	67.4/87.5	1.66	1.48
	SGT	66.7/86.2	67.4/87.5	1.48	1.24
3	RDHS	67.9/88.1	68.1/88.2	1.96	1.73
	SGT	66.3/85.6	68.9/87.9	1.74	1.45

REFERENCES

[1] Zhaowei Cai, Xiaodong He, Jian Sun, and Nuno Vasconcelos. 2017. Deep Learning with Low Precision by Half-wave Gaussian Quantization. In *CVPR*.
[2] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2015. BinaryConnect: Training Deep Neural Networks with binary weights during propagations. arXiv:1511.00363 [cs.LG]
[3] L. Deng, G. Hinton, and B. Kingsbury. 2013. New types of deep neural network learning for speech recognition and related applications: an overview. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. 8599–8603. https://doi.org/10.1109/ICASSP.2013.6639344
[4] Yukun Ding, Jinglan Liu, Jinjun Xiong, and Yiyu Shi. 2019. On the Universal Approximability and Complexity Bounds of Quantized ReLU Neural Networks. In *International Conference on Learning Representations*. https://openreview.net/forum?id=SJe9rh0cFX
[5] Ruihao Gong, Xianglong Liu, Shenghu Jiang, Tianxiang Li, Peng Hu, Jiazhen Lin, Fengwei Yu, and Junjie Yan. 2019. Differentiable Soft Quantization: Bridging Full-Precision and Low-Bit Neural Networks. arXiv:1908.05033 [cs.CV]
[6] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the Knowledge in a Neural Network. arXiv:1503.02531 [stat.ML]
[7] Lu Hou, Quanming Yao, and James T. Kwok. 2017. Loss-aware Binarization of Deep Networks. In *International Conference on Learning Representations*.
[8] Eric Jang, Shixiang Gu, and Ben Poole. 2017. Categorical Reparameterization with Gumbel-Softmax. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=rkE3y85ee
[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25*. 1097–1105.
[10] Fengfu Li, Bo Zhang, and Bin Liu. 2016. Ternary Weight Networks. arXiv:1605.04711 [cs.CV]
[11] Sheng Lin, Xiaolong Ma, Shaokai Ye, Geng Yuan, Kaisheng Ma, and Yanzhi Wang. 2019. Toward Extremely Low Bit and Lossless Accuracy in DNNs with Progressive ADMM. arXiv:1905.00789 [cs.LG]
[12] Xiaofan Lin, Cong Zhao, and Wei Pan. 2017. Towards Accurate Binary Convolutional Neural Network. In *NIPS*.
[13] Jonathan Long, Evan Shelhamer, and Trevor Darrell. 2014. Fully Convolutional Networks for Semantic Segmentation. arXiv:1411.4038 [cs.CV]
[14] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks. arXiv:1603.05279 [cs.CV]
[15] O. Ronneberger, P.Fischer, and T. Brox. 2015. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, Vol. 9351. Springer, 234–241.
[16] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. 2016. Learning structured sparsity in deep neural networks. In *Advances in neural information processing systems*. 2074–2082.
[17] Dongqing Zhang, Jiaolong Yang, Dongqiangzi Ye, and Gang Hua. 2018. LQ-Nets: Learned Quantization for Highly Accurate and Compact Deep Neural Networks. In *European Conference on Computer Vision (ECCV)*.
[18] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. 2017. Incremental Network Quantization: Towards Lossless CNNs with Low-Precision Weights. In *International Conference on Learning Representations, ICLR2017*.
[19] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. 2016. DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients. arXiv:1606.06160 [cs.NE]
[20] Chenzhuo Zhu, Song Han, Huizi Mao, and William J. Dally. 2016. Trained Ternary Quantization. arXiv:1612.01064 [cs.LG]