Compacting Binary Neural Networks by Sparse Kernel Selection

Yikai Wang[1] Wenbing Huang[2] Yinpeng Dong[1,3] Fuchun Sun[1] Anbang Yao[4]
[1]BNRist Center, State Key Lab on Intelligent Technology and Systems,
Department of Computer Science and Technology, Tsinghua University
[2]Gaoling School of Artificial Intelligence, Renmin University of China [3]RealAI [4]Intel Labs China
{yikaiw,fcsun,dongyinpeng}@tsinghua.edu.cn, hwenbing@126.com, anbang.yao@intel.com

Abstract

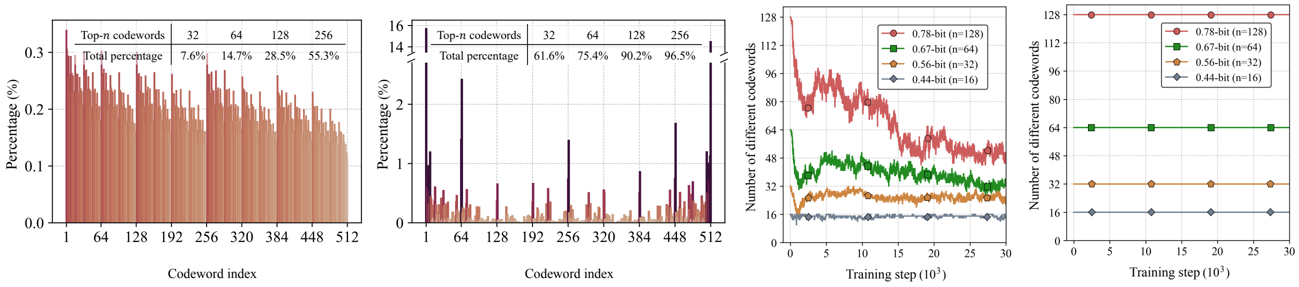
Binary Neural Network (BNN) represents convolution weights with 1-bit values, which enhances the efficiency of storage and computation. This paper is motivated by a previously revealed phenomenon that the binary kernels in successful BNNs are nearly power-law distributed: their values are mostly clustered into a small number of codewords. This phenomenon encourages us to compact typical BNNs and obtain further close performance through learning non-repetitive kernels within a binary kernel subspace. Specifically, we regard the binarization process as kernel grouping in terms of a binary codebook, and our task lies in learning to select a smaller subset of codewords from the full codebook. We then leverage the Gumbel-Sinkhorn technique to approximate the codeword selection process, and develop the Permutation Straight-Through Estimator (PSTE) that is able to not only optimize the selection process end-to-end but also maintain the non-repetitive occupancy of selected codewords. Experiments verify that our method reduces both the model size and bit-wise computational costs, and achieves accuracy improvements compared with state-of-the-art BNNs under comparable budgets.

1. Introduction

It is crucial to design compact Deep Neural Networks (DNNs) which allow the model deployment on resource-constrained embedded devices, since most powerful DNNs including ResNets [10] and DenseNets [13] are storage costly with deep and rich building blocks piled up. Plenty of approaches have been proposed to compress DNNs, among which network quantization [15, 43, 45] is able to reduce memory footprints as well as accelerate the inference speed by converting full-precision weights to discrete values. Binary Neural Networks (BNNs) [2, 14] belong to the family of network quantization but they further constrict the parameter representations to binary values (± 1). In this way, the model is largely compressed. More importantly, floating-point additions and multiplications in conventional DNNs are less required and mostly reduced to bit-wise operations that are well supported by fast inference accelerators [32], particularly when activations are binarized as well. To some extent, this makes BNNs more computationally efficient than other compression techniques, *e.g.*, network pruning [9, 11, 25] and switchable models [37, 41, 42].

Whilst a variety of methods are proposed to improve the performance of BNNs, seldom is there a focus on discussing how the learnt binary kernels are distributed in BNNs. A recent work SNN [38] demonstrates that, by choosing typical convolutional BNN models [27, 31, 32] well trained on ImageNet and displaying the distribution of the 3×3 kernels along all possible $2^{3\times3}$ binary values (*a.k.a.* codewords), these kernels nearly obey the power-law distribution: only a small portion of codewords are activated for the most time. Such a phenomenon is re-illustrated in Figure 1(b). This observation motivates SNN to restrict the size of the codebook by removing those hardly-selected codewords. As a result, SNN is able to compact BNN further since indexing the kernels with a smaller size of codebook results in a compression ratio of $\log_2(n)/\log_2(N)$, where n and N are separately the sizes of the compact and full codebooks.

However, given that the size of codebook is limited (only 512), the sub-codebook degenerates during training since codewords are likely to become repetitive. Therefore, we believe the clustering property of kernels can be further exploited during the training of BNNs. To do so, we reformulate the binary quantization process as a grouping task that selects, for each kernel, the nearest codeword from a binary sub-codebook which is obtained by selecting optimal codewords from the full one. To pursue an optimal solution and retain the non-repetitive occupancy of the selected codewords, we first convert the sub-codebook selection problem to a permutation learning task. However, learning the permutation matrix is non-differential since the permutation matrix is valued with only 0/1 entries. Inspired by the idea in [29], we introduce the Gumbel-Sinkhorn op-



(a) Codebook constructed with sub-vectors (multi-channel codewords)

(b) Codebook constructed with kernels (single-channel codewords) **(ours)**

(c) Product quantization-based optimization for binary codewords

(d) Selection-based optimization for binary codewords **(ours)**

Figure 1. Codebook distributions under different decomposition approaches. Statistics in both sub-figures are collected from the same BNN model (XNOR-Net [32] upon ResNet-18 [10]) well-trained on ImageNet [4]. In (a), each codeword is a flattened sub-vector (of size 1×9). In (b), each codeword is a 3×3 convolution kernel. The codebook in either sub-figure consists of $2^9 = 512$ different codewords. Upper tables provide the total percentages of the top-n most frequent codewords. In (c), we observe that the sub-codebook highly degenerates during training, since codewords tend to be repetitive when being updated independently. While in (d), the diversity of codewords preserves, which implies the superiority of our selection-based learning.

eration to generate a continuous and differential approximation of the permutation matrix. During training, we further develop Permutation Straight-Through Estimator (PSTE), a novel method that tunes the approximated permutation matrix end-to-end while maintaining the binary property of the selected codewords. The details are provided in § 3.2 and § 3.3. We further provide the complexity analysis in § 3.4.

Extensive results on image classification and object detection demonstrate that our architecture noticeably reduces the model size as well as the computational burden. For example, by representing ResNet-18 with 0.56-bit per weight on ImageNet, our method brings in 214× saving of bit-wise operations, and 58× reduction of the model size. Though state-of-the-art BNNs have achieved remarkable compression efficiency, we believe that further compacting BNNs is still beneficial, by which we can adopt deeper, wider, and thus more expressive architectures without exceeding the complexity budget than BNNs. For example, our 0.56-bit ResNet-34 obtains 1.7% higher top-1 accuracy than the state-of-the-art BNN on ResNet-18, while its computational costs are lower and the storage costs are almost the same.

Existing methods [20, 21] (apart from SNN [38]) that also attempt to obtain more compact models than BNNs are quite different with ours as will be described in § 2. One of the crucial points is that their codewords are sub-vectors from (flattened) convolution weights across multiple channels, whereas our each codeword corresponds to a complete kernel that maintains the spatial dimensions (weight and height) of a single channel. The reason why we formulate the codebook in this way stems from the observation in Figure 1(b), where the kernels are sparsely clustered. Differently, as shown in Figure 1(a), the codewords are nearly uniformly activated if the codebook is constructed from flattened sub-vectors, which could because the patterns of the input are spatially selective but channel-wise uniformly distributed. It is hence potential that our method may recover better expressivity of BNNs by following this natural characteristic. In addition, we optimize the codewords via non-repetitive selection from a fixed codebook, which rigorously ensures the dissimilarity between every two codewords and thus enables more capacity than the product quantization method used in [20], as compared in Figure 1(c)(d). On ImageNet with the same backbone, our method exceeds [20] and [21] by 6.6% and 4.5% top-1 accuracies, respectively.

2. Related Work

BNNs. Network quantization methods [6,15,43,45] convert network weights to low-bit values and are appealing for resource-limited devices given the superiority in efficiency. As an extreme solution of quantization, BNNs [2,14,22,36] represent weights and activations with 1-bit (± 1) values, bringing 32× storage compression ratio and 58× practical computational reduction on CPU as reported by [32]. BNNs usually adopt a non-differentiable sign function during the forward pass and the Straight-Through Estimator (STE) [2] for gradient back-propagation. Many attempts are proposed to narrow the performance gap between BNNs and their real-valued counterparts. XNOR-Net [32] adopts floating-point parameters as scaling factors to reduce the quantization error. Bi-Real [26] proposes to add ResNet-like shortcuts to reduce the information loss during binarization. ABC-Net [24] linearly combines multiple binary weight bases to further approximate full-precision weights. ReActNet [27] generalizes activation functions to capture the distribution reshape and shift. New architectures for BNNs can be searched [33] or designed [3] to further improve the trade-off between performance and efficiency.

Compacting BNNs. Our work focuses on an orthogonal venue and investigates how to compact BNNs further. Previously, SNN [38] reveals that binary kernels learnt at convolutional layers of a BNN model are likely to be distributed over kernel subsets. Based on this, SNN randomly samples layer-specific binary kernel subsets and refines them during training. However, the optimization of SNN is easy to attain repetitive binary kernels, *i.e.*, degenerated subsets, leading to a noticeable performance drop compared with conventional BNN. Another method sharing a similar motivation with us is the fractional quantization (FleXOR) [21], which encrypts the sub-vectors of flattened weights to low-dimensional binary codes. Yet, FleXOR cannot track which weights share the same encrypted code, and thus it is needed to decrypt the compressed model back to the full BNN for inference. In our method, the reconstruction of the corresponding BNN is unnecessary, since the computation can be realized in the space of codewords, leading to further reduction of bit-wise computations as detailed in § 3.4. Another research close to our paper is SLBF [20] that applies the idea of stacking low-dimensional filters [40] and the product quantization [7, 17, 35] to BNNs. Similar to [21], this method splits the (flattened) weights into sub-vectors as codewords along the channel direction. As already demonstrated in § 1, our method leverages kernel-wise codebook creation by selection-based codeword optimization, yielding much lower quantization errors than [20, 21].

3. Sparse Kernel Selection

In this section, we introduce how to compact and accelerate BNN further by Sparse Kernel Selection, abbreviated as **Sparks**. Towards this goal, we first formulate the quantization process as grouping convolution kernels into a certain binary codebook. We then show that a more compact sub-codebook can be learnt end-to-end via Gumbel-Sinkhorn ranking. To enable the optimization of the ranking while keeping the binary property, we further propose the Permutation Straight-Through Estimator (PSTE) technique with the convergence analysis. Finally, we contrast the complexity of the model with BNN, and demonstrate that our method is able to not only compress BNN further but also accelerate the speed of BNN during inference.

3.1. Binarization below 1-Bit

Prior to going further, we first provide the notations and necessary preliminaries used in our paper. Let $\boldsymbol{W} \in \mathbb{R}^{C_{\text{out}}\times C_{\text{in}}\times K\times K}$ be the convolution weights, where C_{out} and C_{in} are the numbers of the output and input channels, respectively, and K is the kernel size. As discussed in § 1, we are interested in the quantization for each specific kernel, denoted with a lowercase letter as $\boldsymbol{w} \in \mathbb{R}^{K\times K}$. The quantization process aims to map full-precision weights to a smaller set of discrete finite values. Specifically, we conduct the function sign for each kernel, resulting in $\hat{\boldsymbol{w}} = \text{sign}(\boldsymbol{w}) \in \mathbb{B}$ where $\mathbb{B} = \{-1, +1\}^{K\times K}$ is the set consisting of 1-bit per weight. In what follows, $\mathbb{B}$ is called the (full) codebook, and each element in $\mathbb{B}$ is a codeword.

Generally, the quantization can be rewritten as an optimization problem $\hat{\boldsymbol{w}} = \arg\min_{\boldsymbol{u}\in\mathbb{B}} \|\boldsymbol{u} - \boldsymbol{w}\|_2$ that grouping each kernel $\boldsymbol{w}$ to its nearest codeword in $\mathbb{B}$, where $\|\cdot\|_2$ denotes the ℓ_2 norm. We state this equivalence in the form below, and the proof is provided in Appendix.

Property 1 *We denote $\mathbb{B} = \{-1, +1\}^{K\times K}$ as the codebook of binary kernels. For each $\boldsymbol{w} \in \mathbb{R}^{K\times K}$, the binary kernel $\hat{\boldsymbol{w}}$ can be derived by a grouping process:*

$$\hat{\boldsymbol{w}} = \text{sign}(\boldsymbol{w}) = \arg\min_{\boldsymbol{u}\in\mathbb{B}} \|\boldsymbol{u} - \boldsymbol{w}\|_2. \tag{1}$$

Since the codebook size $|\mathbb{B}| = 2^{K\times K}$, the memory complexity of BNN is equal to $K \times K$. Given Equation 1, one may want to know if we can further reduce the complexity of BNN by, for example, sampling a smaller subset of the codebook $\mathbb{B}$ to replace $\mathbb{B}$ in Equation 1. This is also motivated by Figure 1(b) where the learnt kernels of BNNs are sparsely clustered into a small number of codewords. In this way, each kernel is represented below K^2-bits and thus averagely, each weight is represented less than 1-bit. We thus recast the grouping as

$$\hat{\boldsymbol{w}} = \arg\min_{\boldsymbol{u}\in\mathbb{U}} \|\boldsymbol{u} - \boldsymbol{w}\|_2, \ \textit{s.t.}\ \mathbb{U} \subseteq \mathbb{B}. \tag{2}$$

We denote $|\mathbb{U}| = n$ and $|\mathbb{B}| = N$. By Equation 2, each binary kernel occupies $\log_2(n)$ bits as it can be represented by an index in $\{1, 2, \cdots, n\}$. Thus we obtain a compression ratio of $\log_2(n)/\log_2(N)$.

Different choice of the sub-codebook $\mathbb{U}$ from $\mathbb{B}$ potentially delivers different performance. How can we determine the proper selection we prefer? One possible solution is to optimize codewords and kernels simultaneously by making use of the product quantization method [20, 35]. Nevertheless, this method updates each codeword independently and is prone to deriving repetitive codewords if the optimization space is constrained (as values are limited to ±1). As a consequence, it will hinder the diversity of the codebook and limit the expressivity of the model, which will be verified in § 4.2. Another straightforward method would be sampling the most frequent n codewords from a learnt target BNN (depicted in Figure 1(b)). Yet, such a solution is suboptimal since it solely depends on the weight distribution of the eventual model without the involvement of the specific training dynamics. In the following subsections, we will propose to tackle the sub-codebook selection via an end-to-end approach while retaining the non-repeatability of the codewords.

3.2. Sub-Codebook Selection via Permutation

We learn a permutation of codewords in $\mathbb{B}$ according to their effects on the target loss function, so that the selection of the first n codewords is able to optimize the final performance of the target task. The designed permutation learning keeps the binary property of the selected codewords.

For convenience, we index codewords in $\mathbb{B}$ as a matrix column by column, formulated as $\boldsymbol{B} = [\boldsymbol{u}_1; \boldsymbol{u}_2; \cdots; \boldsymbol{u}_N] \in \{\pm 1\}^{K^2 \times N}$, where each codeword in $\mathbb{B}$ is flattened to a K^2-dimensional vector. Similarly, we convert $\mathbb{U}$ as $\boldsymbol{U} = [\boldsymbol{u}_{s_1}; \boldsymbol{u}_{s_2}; \cdots, \boldsymbol{u}_{s_n}] \in \{\pm 1\}^{K^2 \times n}$, where $s_i \in \{1, 2, \cdots, N\}$ is the index of the i-th selected codeword. We denote the selection matrix as $\boldsymbol{V} \in \{0, 1\}^{N \times n}$, then

$$\boldsymbol{U} = \boldsymbol{B}\boldsymbol{V}, \tag{3}$$

where the entries of $\boldsymbol{V}$ satisfy $\boldsymbol{V}_{s_i,i} = 1$ for $i = 1, \cdots, n$ and are zeros otherwise. The selection by Equation 3 is permutation-dependent; in other words, if we permute the element of $\mathbb{B}$, we may obtain different $\mathbb{U}$. Hence, how to select $\mathbb{U}$ becomes how to first permute $\mathbb{B}$ and then output $\mathbb{U}$ by Equation 3. We denote $\mathbb{P}_N$ the set of N-dimensional permutation matrices: $\mathbb{P}_N = \{\boldsymbol{P} \in \{0, 1\}^{N \times N} \,|\, \boldsymbol{P}\boldsymbol{1}_N = \boldsymbol{1}_N, \boldsymbol{P}^\top \boldsymbol{1}_N = \boldsymbol{1}_N\}$, where $\boldsymbol{1}_N$ is an N-dimensional column vector of ones. The optimization problem in Equation 2 is transformed into

$$\hat{\boldsymbol{w}} = \arg\min_{\boldsymbol{u} \in \mathbb{U}} \|\boldsymbol{u} - \boldsymbol{w}\|_2, \ \textit{s.t.}\ \boldsymbol{U} = \boldsymbol{B}\boldsymbol{P}\boldsymbol{V}, \boldsymbol{P} \in \mathbb{P}_N, \tag{4}$$

where $\boldsymbol{V}$ is fixed as a certain initial selection.

Now, the goal is how to determine a proper permutation matrix $\boldsymbol{P}$. Basically, we can design a neural network to output $\boldsymbol{P}$, and then embed it into the off-the-shelf CNN for the downstream task. Unfortunately, this pipeline fails as the permutation matrix $\boldsymbol{P}$ is discrete, whose values are occupied with 0 or 1, making the permutation network non-differentiable. Joining the recent advancement of permutation learning, we leverage the method proposed by [1] that approximates the permutation matrix by its continuous and differentiable relaxation—the Sinkhorn operator [34].

Given a matrix $\boldsymbol{X} \in \mathbb{R}^{N \times N} (N = |\mathbb{B}|)$, the Sinkhorn operator over $\mathcal{S}(\boldsymbol{X})$ is proceeded as follow,

$$\mathcal{S}^0(\boldsymbol{X}) = \exp(\boldsymbol{X}), \tag{5}$$

$$\mathcal{S}^k(\boldsymbol{X}) = \mathcal{T}_c\left(\mathcal{T}_r(\mathcal{S}^{k-1}(\boldsymbol{X}))\right), \tag{6}$$

$$\mathcal{S}(\boldsymbol{X}) = \lim_{k \to \infty} \mathcal{S}^k(\boldsymbol{X}), \tag{7}$$

where $\mathcal{T}_r(\boldsymbol{X}) = \boldsymbol{X} \oslash (\boldsymbol{X}\boldsymbol{1}_N\boldsymbol{1}_N^\top)$ and $\mathcal{T}_c(\boldsymbol{X}) = \boldsymbol{X} \oslash (\boldsymbol{1}_N\boldsymbol{1}_N^\top\boldsymbol{X})$ are the row-wise and column-wise normalization operators, and $\oslash$ denotes the element-wise division. For stability purpose, both normalization operators are calculated in the log domain in practice. The work by [34] proved that $\mathcal{S}(\boldsymbol{X})$ belongs to the Birkhoff polytope—the set of doubly stochastic matrices.

Through adding a temperature τ, it can be proved that $\lim_{\tau \to 0^+} \mathcal{S}(\boldsymbol{X}/\tau) = \arg\max_{\boldsymbol{P} \in \mathbb{P}_N} \|\boldsymbol{P} - \boldsymbol{X}\|_2$ holds almost surely [29]. It means we obtain an approximated permutation matrix $\mathcal{S}^k(\boldsymbol{X})$ (that is closest to $\boldsymbol{X}$) with sufficiently large k and small τ. Inspired by [16], we add a Gumbel noise to make the result follow the Gumbel-Matching distribution $\mathcal{G.M.}(\boldsymbol{X})$, namely, $\mathcal{S}^k((\boldsymbol{X} + \epsilon)/\tau)$, where ϵ is sampled from standard i.i.d. Gumbel distribution.

By substituting the Gumbel-Sinkhorn matrix into Equation 3, we characterize the sub-codebook selection as

$$\boldsymbol{U} = \boldsymbol{B}\mathcal{S}^k((\boldsymbol{X} + \epsilon)/\tau)\boldsymbol{V}, \tag{8}$$

where $\boldsymbol{V}$ is fixed as a certain initial selection as mentioned, $\boldsymbol{X}$ is regarded as a learnable parameter, k and τ are hyper-parameters. For $\boldsymbol{V}$, we can simply let the entries to be zeros unless $\boldsymbol{V}_{i,i} = 1$, where $i = 1, \cdots, n$, which indicates selecting the first n columns from $\boldsymbol{B}\mathcal{S}^k((\boldsymbol{X} + \epsilon)/\tau)$.

3.3. Learning by PSTE

Recalling that both k and τ are finitely valued, the Gumbel-Sinkhorn matrix $\boldsymbol{P}_{\text{GS}} = \mathcal{S}^k((\boldsymbol{X} + \epsilon)/\tau)$ is not strictly a permutation matrix with 0/1 entries. This will violate the binary property of $\boldsymbol{U}$ by Equation 8, making the binarization of Equation 2 meaningless. To address this issue, we derive the exact permutation matrix $\boldsymbol{P}_{\text{real}}$ of $\boldsymbol{P}_{\text{GS}}$ by making use of the Hungarian algorithm [30] during the forward pass. By treating the $\boldsymbol{P}_{\text{GS}}$ as a reward matrix, deriving $\boldsymbol{P}_{\text{real}}$ becomes an assignment problem that can be solved by the Hungarian method in polynomial time. We summarize the forward update of the convolution kernel $\boldsymbol{w}_c \in \mathbb{R}^{K^2}$ for each input and output channel as follow,

$$\boldsymbol{P}_{\text{real}} = \text{Hungarian}(\boldsymbol{P}_{\text{GS}}), \tag{9}$$

$$\boldsymbol{U} = \boldsymbol{B}\boldsymbol{P}_{\text{real}}\boldsymbol{V}, \tag{10}$$

$$\hat{\boldsymbol{w}}_c = \arg\min_{\boldsymbol{u} \in \mathbb{U}} \|\boldsymbol{u} - \boldsymbol{w}_c\|_2, \tag{11}$$

where $\text{Hungarian}(\cdot)$ denotes the Hungarian algorithm.

In the backward pass, we transfer the gradient of the exact permutation matrix directly to the Gumbel-Sinkhorn matrix. This is inspired by the Straight-Through Estimator (STE) technique [2] in previous literature. We call our method PSTE for its specification to permutation learning here. The backward pass is depicted below,

$$\mathbf{g}(\boldsymbol{w}_{c,i}) \approx \begin{cases} \mathbf{g}(\hat{\boldsymbol{w}}_{c,i}), & \text{if } \boldsymbol{w}_{c,i} \in (-1, 1), \\ 0, & \text{otherwise}, \end{cases} \tag{12}$$

$$\mathbf{g}(\boldsymbol{u}_j) = \sum_{c=1}^{C_{\text{in}} \times C_{\text{out}}} \mathbf{g}(\hat{\boldsymbol{w}}_c) \cdot \mathbb{I}_{\boldsymbol{u}_j = \arg\min_{\boldsymbol{u} \in \mathbb{U}} \|\boldsymbol{u} - \boldsymbol{w}_c\|_2}, \tag{13}$$

$$\mathbf{g}(\boldsymbol{P}_{\text{real}}) = \boldsymbol{B}^\top \mathbf{g}(\boldsymbol{U})\boldsymbol{V}^\top, \tag{14}$$

$$\mathbf{g}(\boldsymbol{P}_{\text{GS}}) \approx \mathbf{g}(\boldsymbol{P}_{\text{real}}), \tag{15}$$

where $\mathbf{g}(\cdot)$ computes the gradient. $\boldsymbol{w}_{c,i}$ and $\hat{\boldsymbol{w}}_{c,i}$ denote the i-th entries of $\boldsymbol{w}_c$ and $\hat{\boldsymbol{w}}_c$, respectively, with $i = 1, 2, \cdots, K^2$. $\mathbb{I}_{\{\cdot\}}$ defines the indicator function. Particularly, Equation 12 follows the idea of STE and Equation 13 assigns the gradient of the binary weight to its nearest codeword. In practice, all forward and backward passes can

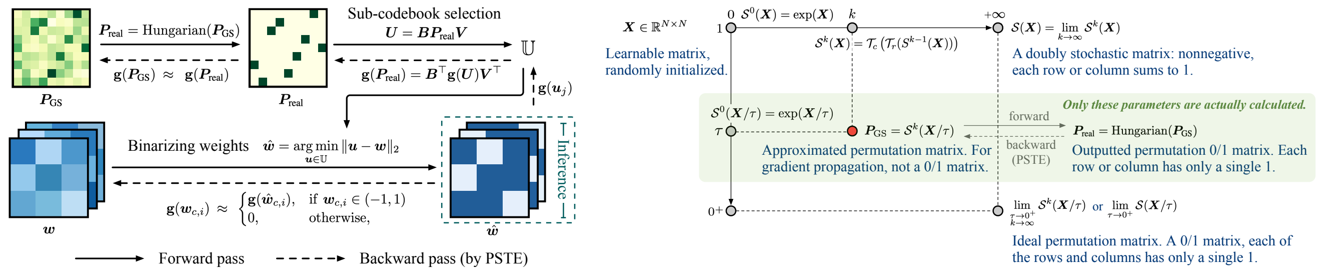

$P_{\text{real}} = \text{Hungarian}(P_{\text{GS}})$
$g(P_{\text{GS}}) \approx g(P_{\text{real}})$
P_{GS}
P_{real}
Sub-codebook selection
$U = BP_{\text{real}}V$
$g(P_{\text{real}}) = B^\top g(U)V^\top$
$\mathbb{U}$
$g(u_j)$
Binarizing weights $\hat{w} = \arg\min_{u\in\mathbb{U}} \|u - w\|_2$
$g(w_{c,i}) \approx \begin{cases} g(\hat{w}_{c,i}), & \text{if } w_{c,i} \in (-1,1) \\ 0, & \text{otherwise,} \end{cases}$
w
$\hat{w}$
Inference
Forward pass
Backward pass (by PSTE)
$X \in \mathbb{R}^{N\times N}$
Learnable matrix, randomly initialized.
0 $\mathcal{S}^0(X) = \exp(X)$ k $+\infty$
1
$\mathcal{S}^k(X) = \mathcal{T}_c(\mathcal{T}_r(\mathcal{S}^{k-1}(X)))$
$\mathcal{S}(X) = \lim_{k\to\infty} \mathcal{S}^k(X)$
A doubly stochastic matrix: nonnegative, each row or column sums to 1.
Only these parameters are actually calculated.
$\mathcal{S}^0(X/\tau) = \exp(X/\tau)$
τ
$P_{\text{GS}} = \mathcal{S}^k(X/\tau)$
Approximated permutation matrix. For gradient propagation, not a 0/1 matrix.
forward
backward (PSTE)
$P_{\text{real}} = \text{Hungarian}(P_{\text{GS}})$
Outputted permutation 0/1 matrix. Each row or column has only a single 1.
$0+$
$\lim_{\substack{\tau\to 0^+ \\ k\to\infty}} \mathcal{S}^k(X/\tau)$ or $\lim_{\tau\to 0^+} \mathcal{S}(X/\tau)$
Ideal permutation matrix. A 0/1 matrix, each of the rows and columns has only a single 1.

Figure 2. **Left:** A schematic overview of the optimization process. Full-precision weights are binarized by grouping to the nearest codeword of the sub-codebook $\mathbb{U}$, which is obtained by Gumbel-Sinkhorn and optimized by PSTE. Forward and backward passes illustrate how the network calculates and updates. **Right**: Relationship of notations for a better understanding. The horizontal axis and vertical axis stand for the values of k and τ, respectively. Only parameters in the green region are actually calculated during training, while the others are only for the understanding purpose. All these parameters are NOT needed during inference.

be implemented by matrix/tensor operations, and thus our method is computationally friendly on GPUs.

An overall framework including the forward and backward processes is illustrated in Figure 2.

Convergence analysis. Besides using STE to update $\boldsymbol{w}$ in Equation 12, we approximate the gradient of the Gumbel-Sinkhorn matrix $\boldsymbol{P}_{\text{GS}}$ with $\boldsymbol{P}_{\text{real}}$ in Equation 15, which, inevitably, will cause variations in training dynamics. Fortunately, we have the following theorem to guarantee the convergence for sufficiently large k and small τ.

Lemma 1 *For sufficiently large k and small τ, we define the entropy of a doubly-stochastic matrix $\boldsymbol{P}$ as $h(\boldsymbol{P}) = -\sum_{i,j} P_{i,j} \log P_{i,j}$, and denote the rate of convergence for the Sinkhorn operator as r $(0 < r < 1)$*[1]*. There exists a convergence series s_τ ($s_\tau \to 0$ when $\tau \to 0^+$) that satisfies*

$$\|\boldsymbol{P}_{\text{real}} - \boldsymbol{P}_{\text{GS}}\|_2^2 = \mathcal{O}\big(s_\tau^2 + r^{2k}\big). \tag{16}$$

Theorem 1 *Assume that the training objective f* w.r.t. $\boldsymbol{P}_{\text{GS}}$ *is L-smooth, and the stochastic gradient of $\boldsymbol{P}_{\text{real}}$ is bounded by $\mathbb{E}\|\mathbf{g}(\boldsymbol{P}_{\text{real}})\|_2^2 \le \sigma^2$. Denote the rate of convergence for the Sinkhorn operator as r $(0 < r < 1)$ and the stationary point as $\boldsymbol{P}_{\text{GS}}^\star$. Let the learning rate of PSTE be $\eta = \frac{c}{\sqrt{T}}$ with $c = \sqrt{\frac{f(\boldsymbol{P}_{\text{GS}}^0) - f(\boldsymbol{P}_{\text{GS}}^\star)}{L\sigma^2}}$. For a uniformly chosen $\boldsymbol{u}$ from the iterates $\{\boldsymbol{P}_{\text{real}}^0, \cdots, \boldsymbol{P}_{\text{real}}^T\}$, concretely $\boldsymbol{u} = \boldsymbol{P}_{\text{real}}^t$ with the probability $p_t = \frac{1}{T+1}$, it holds in expectation over the stochasticity and the selection of $\boldsymbol{u}$:*

$$\mathbb{E}\|\nabla f(\boldsymbol{u})\|_2^2 = \mathcal{O}\left(\sigma\sqrt{\frac{f(\boldsymbol{P}_{\text{GS}}^0) - f(\boldsymbol{P}_{\text{GS}}^\star)}{T/L}} + L^2\big(s_\tau^2 + r^{2k}\big)\right). \tag{17}$$

Note that in Theorem 1, the objective function f could be a non-convex function, which accords with the case when using a neural network. Proofs for Lemma 1 and Theorem 1 are provided in Appendix.

[1]The Sinkhorn operator has a rate of convergence r bounded by a value lower than 1 as proved by [18].

3.4. Complexity Analysis During Inference

Storage. We consider the convolutional layer with 3×3 kernels. In a conventional binary convolutional layer, the weights requires $C_{\text{out}} \times C_{\text{in}} \times K \times K$ bits. For our method, we only store the sub-codebook $\mathbb{U}$ and the index of each kernel by Equation 2. Storing $\mathbb{U}$ needs $n \times K \times K$ bits, where $n = |\mathbb{U}|$. Since $n \le N = 2^{K\times K} \ll C_{\text{out}} \times C_{\text{in}}$ for many popular CNNs (*e.g.*, ResNets [10]), particularly if all layers share the same $\mathbb{U}$ which is the case of our implementation, only the indexing process counts majorly. The indexing process needs $\log_2(n)$ bits for a kernel, hence indexing all kernels takes $C_{\text{out}} \times C_{\text{in}} \times \log_2(n)$. As a result, the ratio of storage saving by our method is $\log_2(n)/(K \times K) = \log_2(n)/\log_2(N) \le 1$ compared to a conventional BNN.

Computation. We adopt BOPs to measure the computational costs and follow the calculation method in [26–28,32] where each BOP represents two bit-wise operations. In a conventional BNN, the convolution operation between the input feature maps ($C_{\text{in}} \times H \times W$) and weights ($C_{\text{out}} \times C_{\text{in}} \times K \times K$) takes $\text{BOPs} = H \times W \times C_{\text{in}} \times K^2 \times C_{\text{out}}$, where H and W are the height and width of the feature map, respectively. For our method, the kernel grouping process implies that some weights will share the same value, which enables us to further reduce BOPs. To do so, we pre-calculate the convolutions between the input feature maps ($C_{\text{in}} \times H \times W$) and each codeword ($K \times K$). For facility, we reshape the codeword as $1 \times C_{\text{in}} \times K \times K$ by repeating the second dimension over C_{in} times. Then, the pre-convolution for all codewords gives rise to a tensor $\mathcal{T}$ ($n \times C_{\text{in}} \times H \times W$) and costs $\text{BOPs}_1 = H \times W \times C_{\text{in}} \times K^2 \times n$. We reconstruct the convolution result between the input feature maps and the convolution weights given the pre-calculated tensor $\mathcal{T}$. Specifically, for each output channel, we query indices of the input channels *w.r.t.* $\mathbb{U}$, collect feature maps from $\mathcal{T}$ according to the indices, and sum the feature maps as the final result. This process consumes $\text{BOPs}_2 = C_{\text{out}} \times (C_{\text{in}} \times H \times W - 1)/2$. Therefore, Sparks needs $\text{BOPs}_1 + \text{BOPs}_2$ which is far less than BOPs when $K = 3$, as $n < C_{out}$ in general.

Method **(ResNet-18)**	Bit-width (W/A)	Accuracy Top-1 (%)	Storage (Mbit)	BOPs ($\times 10^8$)	Method **(VGG-small)**	Bit-width (W/A)	Accuracy Top-1 (%)	Storage (Mbit)	BOPs ($\times 10^8$)
Full-precision	32/32	94.8	351.5 (1×)	350.3 (1×)	Full-precision	32/32	94.1	146.2 (1×)	386.6 (1×)
XNOR-Net [32]	1/1	90.2	11.0 (32×)	5.47 (64×)	XNOR-Net [32]	1/1	89.8	4.57 (32×)	6.03 (64×)
Bi-RealNet [26]	1/1	90.2	11.0 (32×)	5.47 (64×)	LAB [12]	1/1	87.7	4.57 (32×)	6.03 (64×)
RAD [5]	1/1	90.5	11.0 (32×)	5.47 (64×)	RAD [5]	1/1	90.0	4.57 (32×)	6.03 (64×)
IR-Net [31]	1/1	91.5	11.0 (32×)	5.47 (64×)	IR-Net [31]	1/1	90.4	4.57 (32×)	6.03 (64×)
RBNN [23]	1/1	92.2	11.0 (32×)	5.47 (64×)	RBNN [23]	1/1	91.3	4.57 (32×)	6.03 (64×)
ReActNet [27]	1/1	92.3	11.0 (32×)	5.47 (64×)	SLB [39]	1/1	92.0	4.57 (32×)	6.03 (64×)
SLBF [20]	0.55/1	89.3 ±0.5	6.05 (58×)	2.94 (119×)	SLBF [20]	0.53/1	89.4 ±0.4	2.42 (60×)	3.17 (122×)
FleXOR [21]	0.80/1	90.9 ±0.2	8.80 (40×)	5.47 (64×)	FleXOR [21]	0.80/1	90.6 ±0.1	3.66 (40×)	6.03 (64×)
FleXOR [21]	0.60/1	89.8 ±0.3	6.60 (53×)	5.47 (64×)	FleXOR [21]	0.60/1	89.2 ±0.2	2.74 (53×)	6.03 (64×)
Sparks (ours)	0.78/1	92.2 ±0.1	8.57 **(41×)**	3.96 **(88×)**	Sparks (ours)	0.78/1	91.7 ±0.2	3.55 **(41×)**	3.46 **(112×)**
Sparks (ours)	0.67/1	92.0 ±0.2	7.32 **(48×)**	2.97 **(118×)**	Sparks (ours)	0.67/1	91.6 ±0.1	3.05 **(48×)**	1.94 **(199×)**
Sparks (ours)	0.56/1	91.5 ±0.3	6.10 **(58×)**	1.63 **(215×)**	Sparks (ours)	0.56/1	91.3 ±0.3	2.54 **(58×)**	1.13 **(342×)**
Sparks (ours)	0.44/1	90.8 ±0.2	4.88 **(72×)**	0.97 **(361×)**	Sparks (ours)	0.44/1	90.8 ±0.3	2.03 **(72×)**	0.74 **(522×)**

Table 1. Comparisons of top-1 accuracies with state-of-the-art methods on the CIFAR10 dataset.

4. Experiments

Our method is evaluated on two tasks: image classification and object detection (in Appendix). For image classification, we contrast the performance of our Sparks with state-of-the-art (SOTA) methods on CIFAR10 [19] and ImageNet (ILSVRC2012) [4] following the standard data splits.

Implementation. We follow the standard binarization in ReActNet [27] and perform a two-stage training. First, the network is trained from scratch with binarized activations and real-valued weights. Second, the network takes the weights from the first step and both weights and activations are binarized. As suggested by [26, 32], we keep the weights and activations in the first convolutional and the last fully-connected layers to be real-valued. More implementation details (*e.g.*, learning rate, epoch) are in the Appendix.

We speed up our training twice by exploiting the symmetry of the binary kernels: for each codeword in the codebook $|\mathbb{B}|$, its "opposite" term (with opposite signs) is also contained in $|\mathbb{B}|$. Speed-up details are contained in Appendix. Hyper-parameters are $k = 10$ and $\tau = 10^{-2}$ by default.

On ImageNet, Sparks needs 30.2 hours to train ResNet-18 based on 8 V100s, and the BNN baseline [27] needs 24.5 hours. The computation overhead is acceptable. Training Sparks can be easily accelerated as presented in Appendix

Calculations of storage and BOPs savings are based on the measurements used in [26, 32]. Specifically, compared to full-precision convolutional layers with 32-bit weights, using 1-bit weights and activations gains up to a $\sim$32× storage saving; in addition, the convolution operation could be implemented by the bit-wise xnor operation followed by a popcount operation, which leads to a $\sim$64× computational saving. Throughout our results, we provide the amount of bit-wise parameters in all binarized layers as the storage.

4.1. Comparisons with SOTA Methods

Evaluation on CIFAR10. Table 1 provides the performance comparisons with SOTA BNNs on CIFAR10. Moreover, SLBF [20] and FleXOR [21] that derive weights below 1-bit are re-implemented upon the same backbone (ReActNet) and same binarization settings (both weights and activations are binarized) as our method for a fair comparison. By setting n to 32, 64, and 128, we obtain networks of 0.56-bit, 0.67-bit, and 0.78-bit, respectively. Clearly, our approach is able to achieve accuracies close to standard BNNs with a much lower cost of storage and BOPs on both ResNet-18 and VGG-small. FleXOR also compacts BNNs but delivers no BOPs reduction. SLBF reduces both storage and BOPs but suffers from more accuracy drops as it conducts convolution in the space of multi-channel codewords, leading to much larger quantization errors, as previously described in Figure 1. Our Sparks remarkably outperforms SLBF and FleXOR under almost the same compression level of storage (*e.g.* for ResNet-18, our 0.56-bit gains accuracy 91.5%, while 0.60-bit FleXOR and 0.55-bit SLBF yield 89.8% and 89.3%, respectively), which verify the effectiveness of our proposed method.

Evaluation on ImageNet. In Table 2, we compare Sparks with SOTA methods upon ResNet-18 on ImageNet. Consistent with the results on CIFAR10, our method is able to achieve competitive classification accuracy compared to SOTA BNNs, with a dramatic drop in model size and computation time. Under comparable bit-widths, our 0.56-bit method exceeds 0.55-bit SLBF and 0.6-bit FleXOR by 6.6% and 4.5%, respectively, with even fewer BOPs. Thanks to the remarkable benefit in model compression, we can apply our Sparks on wider and deeper models while staying the same complexity budget as BNNs. For example, in Table 3, we follow ABC-Net [24] and apply our

Method	Bit-width (W/A)	Accuracy (%) Top-1	Top-5	Storage (Mbit)	BOPs ($\times 10^9$)
Full-precision	32/32	69.6	89.2	351.5	107.2 (1×)
BNN [14]	1/1	42.2	69.2	11.0 (32×)	1.70 (63×)
XNOR-Net [32]	1/1	51.2	73.2	11.0 (32×)	1.70 (63×)
Bi-RealNet [26]	1/1	56.4	79.5	11.0 (32×)	1.68 (64×)
IR-Net [31]	1/1	58.1	80.0	11.0 (32×)	1.68 (64×)
LNS [8]	1/1	59.4	81.7	11.0 (32×)	1.68 (64×)
RBNN [23]	1/1	59.9	81.9	11.0 (32×)	1.68 (64×)
Ensemble-BNN [44]	(1/1)×6	61.0	-	65.9 (5.3×)	10.6 (10×)
ABC-Net [24]	(1/1)×5^2	65.0	85.9	274.5 (1.3×)	42.5 (2.5×)
Real-to-Bin [28]	1/1	65.4	86.2	11.0 (32×)	1.68 (64×)
ReActNet [27]	1/1	65.9	86.4	11.0 (32×)	1.68 (64×)
SLBF [20]	0.55/1	57.7	80.2	6.05 (58×)	0.92 (117×)
SLBF [20]	0.31/1	52.5	76.1	3.41 (103×)	0.98 (110×)
FleXOR [21]	0.80/1	62.4	83.0	8.80 (40×)	1.68 (64×)
FleXOR [21]	0.60/1	59.8	81.9	6.60 (53×)	1.68 (64×)
Sparks (ours)	0.78/1	65.5	86.2	8.57 **(41×)**	1.22 **(88×)**
Sparks (ours)	0.67/1	65.0	86.0	7.32 **(48×)**	0.88 **(122×)**
Sparks (ours)	0.56/1	64.3	85.6	6.10 **(58×)**	0.50 **(214×)**

Table 2. Comparisons of top-1 and top-5 accuracies with state-of-the-art methods on ImageNet based on ResNet-18. Calculation details for the storage and BOPs are provided in Appendix.

Method	Backbone	Bit-width (W/A)	Accuracy (%) Top-1	Top-5	Storage (Mbit)	BOPs ($\times 10^9$)
ReActNet [27]	ResNet-18	1/1	65.9	86.4	11.0	1.68
Sparks-wide	ResNet-18 (+ABC-Net [24])	(0.56/1)×3	**66.7**	**86.9**	18.3	**1.50**
Sparks-deep	ResNet-34	0.56/1	**67.6**	**87.5**	11.7	**0.96**
Sparks-deep	ResNet-34	0.44/1	**66.4**	**86.7**	**9.4**	**0.58**

Table 3. Results when extending our Sparks to wider or deeper models.

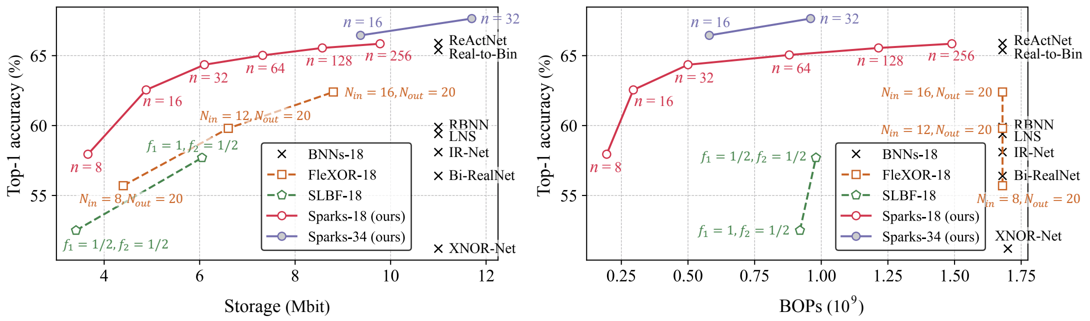


Figure 3. Trade-off between performance and complexity on ImageNet. For all methods, -18 indicates using ResNet-18 as the backbone, and -34 indicates ResNet-34. The symbol n is the sub-codebook size as defined in § 3.1; $N_{in}, N_{out}, f_1, f_2$ are hyper-parameters that control the complexity as defined in FleXOR [21] and SLBF [20].

0.56-bit model on ResNet-18 by using three branches of convolutions and a single branch of activation, denoted as Sparks-wide; we also adopt 0.56-bit and 0.44-bit Sparks on ResNet-34 (almost twice in depth compared to ResNet-18), both denoted as Sparks-deep. We observe that all our variants surpass ReActNet-18 (currently the best ResNet-18-based BNN) with almost the same or even less cost in model complexity. Specifically, our 0.44-bit Sparks-deep defeats ReActNet-18 in accuracy with the least cost of complexity. To better visualize the trade-off between performance and efficiency, we contrast our models against existing methods with varying storage and BOPs in Figure 3. We leave the comparison with SNN [38] in Figure 4, since we re-implemented SNN under a fair two-stage pipeline, which improves the absolute accuracy of SNN by $6.9\% \sim 8.1\%$.

4.2. Ablation Studies

Validity of Gumbel-Sinkhorn. We test the advantage of applying the Gumbel-Sinkhorn technique to codewords selection. Three baselines to construct the sub-codebook are

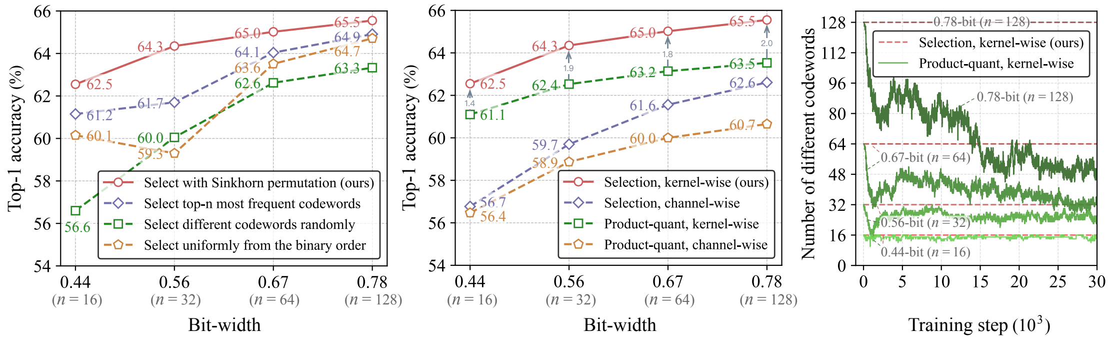


Figure 4. Ablation studies on ImageNet with ResNet-18. All experiments adopt the same baseline for a fair comparison. **Left**: comparisons of different methods to select codewords. **Middle:** kernel-wise vs channel-wise codewords, and selection-based vs product-quantization(quant)-based learning. Note that "Product-quant, kernel-wise" refers to our implementation of SNN [38] under a fair two-stage training framework for improved performance. **Right:** sub-codebook degeneration when learning codewords with product-quantization.

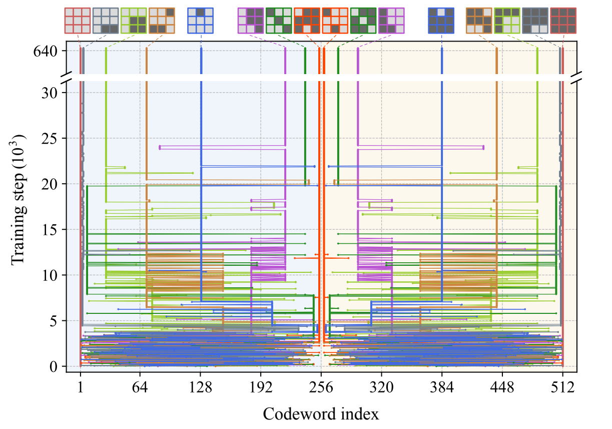


Figure 5. Codewords selection during training where $n = 16$. We have visualized the final selected 16 codewords at the top of the figure, where light gray indicates -1 and dark gray indicates $+1$.

considered: (1) Directly selecting the top-n frequent codewords from a learnt BNN (ReActNet-18). (2) Randomly selecting codewords. (3) Selecting codewords with an equal interval of indices, by setting $s_i = \lfloor \frac{i}{n-1} \rfloor \times (N-1)+1$ in Equation 3. Figure 4 (Left) shows that these baselines are much inferior to ours, indicating the superiority of our permutation learning. We observe a severe performance cutdown at 0.56-bit of the third baseline, reflecting the unpredictable performance variation for naive selection methods.

Comparison with product quantization. Our proposed sub-codebook construction is related with but distinct from the product quantization [20, 35] in two aspects: (1) As already specified in Figure 1, we construct the codebook with kernel-wise codewords rather than channel-wise ones. (2) We learn to jointly select codewords from a fixed codebook instead of directly optimizing the codewords independently, which avoids the degeneration of the sub-codebook diversity. Figure 4 (Middle) illustrates the comparisons regarding these two aspects. We observe that using kernel-wise codewords largely outperforms using the channel-wise codewords, and the selection-based optimization consistently achieves better performance than the product quantization (such as SNN [38]) by a significant margin. In Figure 4 (Right), the diversity of codewords is severely degenerated when applying the product quantization, whereas our method preserves the diversity.

Convergence. Figure 5 displays the codewords selection process during training for 0.44-bit Sparks (ResNet-18) on ImageNet. The selection initially fluctuates but converges in the end. This explains the convergence of our algorithm, which complies with our derivations in Theorem 1.

Practical inference on FPGA. We adopt the hardware framework SystemC-TLM[2] to model an FPGA. Our 0.56-bit Sparks achieves 1.167ms (on ResNet-18) and 2.391ms (on ResNet-34). The corresponding BNNs achieve 3.713ms and 7.806ms. Thus we have over three times acceleration.

5. Conclusion

We propose a novel method named Sparse Kernel Selection (Sparks), which devises below 1-bit models by grouping kernels into a selected sub-codebook. The selection process is learnt end-to-end with Permutation Straight-Through Estimator (PSTE). Experiments show that our method is applicable for general tasks including image classification and object detection. One potential limitation of our research lies in that model compression requires the access of model parameters, leading to the risk of unprotected privacy.

Acknowledgement

This work is funded by the Sino-German Collaborative Research Project Crossmodal Learning (NSFC 62061136001/DFG TRR169). W. Huang, F. Sun, and A. Yao are corresponding authors. Part of this work was done when the first author was an intern at Intel Labs China. Y. Wang and Y. Dong are supported by the Shuimu Tsinghua Scholar Program.

[2]https://www.accellera.org/community/systemc/about-systemc-tlm

References

[1] Ryan Prescott Adams and Richard S. Zemel. Ranking via sinkhorn propagation. In *arXiv preprint arXiv:1106.1925*, 2011. 4

[2] Yoshua Bengio, Nicholas Léonard, and Aaron C. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. In *arXiv preprint arXiv:1308.3432*, 2013. 1, 2, 4

[3] Adrian Bulat, Brais Martínez, and Georgios Tzimiropoulos. High-capacity expert binary networks. In *arXiv preprint arXiv:2010.03558*, 2020. 2

[4] Jia Deng, Wei Dong, Richard Socher, Li Jia Li, Kai Li, and Fei Fei Li. Imagenet: a large-scale hierarchical image database. In *CVPR*, 2009. 2, 6

[5] Ruizhou Ding, Ting-Wu Chin, Zeye Liu, and Diana Marculescu. Regularizing activation distribution for training binarized deep networks. In *CVPR*, 2019. 6

[6] Yinpeng Dong, Renkun Ni, Jianguo Li, Yurong Chen, Hang Su, and Jun Zhu. Stochastic quantization for learning accurate low-bit deep neural networks. In *IJCV*, 2019. 2

[7] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. Optimized product quantization. In *TPAMI*, 2014. 3

[8] Kai Han, Yunhe Wang, Yixing Xu, Chunjing Xu, Enhua Wu, and Chang Xu. Training binary neural networks through learning with noisy supervision. In *ICML*, 2020. 7

[9] Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks. In *NeurIPS*, 2015. 1

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 1, 2, 5

[11] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *ICCV*, 2017. 1

[12] Lu Hou, Quanming Yao, and James T. Kwok. Loss-aware binarization of deep networks. In *ICLR*, 2017. 6

[13] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *CVPR*, 2017. 1

[14] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *NeurIPS*, 2016. 1, 2, 7

[15] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. In *JMLR*, 2017. 1, 2

[16] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *ICLR*, 2017. 4

[17] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. In *TPAMI*, 2011. 3

[18] Philip A. Knight. The sinkhorn-knopp algorithm: Convergence and applications. In *SIAM J. Matrix Anal. Appl.*, 2008. 5

[19] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. In *Tech Report*, 2009. 6

[20] Weichao Lan and Liang Lan. Compressing deep convolutional neural networks by stacking low-dimensional binary convolution filters. In *arXiv preprint arXiv:2010.02778*, 2020. 2, 3, 6, 7, 8

[21] Dongsoo Lee, Se Jung Kwon, Byeongwook Kim, Yongkweon Jeon, Baeseong Park, and Jeongin Yun. Flexor: Trainable fractional quantization. In *NeurIPS*, 2020. 2, 3, 6, 7

[22] Mingbao Lin, Rongrong Ji, Zihan Xu, Baochang Zhang, Fei Chao, Mingliang Xu, Chia-Wen Lin, and Ling Shao. Siman: Sign-to-magnitude network binarization. In *TPAMI*, 2022. 2

[23] Mingbao Lin, Rongrong Ji, Zihan Xu, Baochang Zhang, Yan Wang, Yongjian Wu, Feiyue Huang, and Chia-Wen Lin. Rotated binary neural network. In *NeurIPS*, 2020. 6, 7

[24] Xiaofan Lin, Cong Zhao, and Wei Pan. Towards accurate binary convolutional neural network. In *NeurIPS*, 2017. 2, 6, 7

[25] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *ICCV*, 2017. 1

[26] Zechun Liu, Wenhan Luo, Baoyuan Wu, Xin Yang, Wei Liu, and Kwang-Ting Cheng. Bi-real net: Binarizing deep network towards real-network performance. In *IJCV*, 2020. 2, 5, 6, 7

[27] Zechun Liu, Zhiqiang Shen, Marios Savvides, and Kwang-Ting Cheng. Reactnet: Towards precise binary neural network with generalized activation functions. In *ECCV*, 2020. 1, 2, 5, 6, 7

[28] Brais Martínez, Jing Yang, Adrian Bulat, and Georgios Tzimiropoulos. Training binary neural networks with real-to-binary convolutions. In *ICLR*, 2020. 5, 7

[29] Gonzalo E. Mena, David Belanger, Scott W. Linderman, and Jasper Snoek. Learning latent permutations with gumbel-sinkhorn networks. In *ICLR*, 2018. 1, 4

[30] James Munkres. Algorithms for the assignment and transportation problems. In *Journal of the Society for Industrial and Applied Mathematics*, 1957. 4

[31] Haotong Qin, Ruihao Gong, Xianglong Liu, Mingzhu Shen, Ziran Wei, Fengwei Yu, and Jingkuan Song. Forward and backward information retention for accurate binary neural networks. In *CVPR*, 2020. 1, 6, 7

[32] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *ECCV*, 2016. 1, 2, 5, 6, 7

[33] Mingzhu Shen, Kai Han, Chunjing Xu, and Yunhe Wang. Searching for accurate binary neural architectures. In *ICCV Workshops*, 2019. 2

[34] Richard Sinkhorn. A relationship between arbitrary positive matrices and doubly stochastic matrices. In *The Annals of Mathematical Statistics*, 1964. 4

[35] Pierre Stock, Armand Joulin, Rémi Gribonval, Benjamin Graham, and Hervé Jégou. And the bit goes down: Revisiting the quantization of neural networks. In *ICLR*, 2020. 3, 8

[36] Zhijun Tu, Xinghao Chen, Pengju Ren, and Yunhe Wang. Adabin: Improving binary neural networks with adaptive binary sets. In *ECCV*, 2022. 2

[37] Yikai Wang, Fuchun Sun, Duo Li, and Anbang Yao. Resolution switchable networks for runtime efficient image recognition. In *ECCV*, 2020. 1

[38] Yikai Wang, Yi Yang, Fuchun Sun, and Anbang Yao. Sub-bit neural networks: Learning to compress and accelerate binary neural networks. In *ICCV*, 2021. 1, 2, 7, 8

[39] Zhaohui Yang, Yunhe Wang, Kai Han, Chunjing Xu, Chao Xu, Dacheng Tao, and Chang Xu. Searching for low-bit weights in quantized neural networks. In *NeurIPS*, 2020. 6

[40] Zhaohui Yang, Yunhe Wang, Chuanjian Liu, Hanting Chen, Chunjing Xu, Boxin Shi, Chao Xu, and Chang Xu. Legonet: Efficient convolutional neural networks with lego filters. In *ICML*, 2019. 3

[41] Jiahui Yu and Thomas S. Huang. Universally slimmable networks and improved training techniques. In *ICCV*, 2019. 1

[42] Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas S. Huang. Slimmable neural networks. In *ICLR*, 2019. 1

[43] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. In *arXiv preprint arXiv:1606.06160*, 2016. 1, 2

[44] Shilin Zhu, Xin Dong, and Hao Su. Binary ensemble neural network: More bits per network or more networks per bit? In *CVPR*, 2019. 7

[45] Bohan Zhuang, Chunhua Shen, Mingkui Tan, Lingqiao Liu, and Ian D. Reid. Towards effective low-bitwidth convolutional neural networks. In *CVPR*, 2018. 1, 2