Domain Generalization Capability Enhancement for Binary Neural Networks

Jianming Ye
jmye@pku.edu.cn
Shunan Mao
snmao@pku.edu.cn
Shiliang Zhang
slzhang.jdl@pku.edu.cn

VMC Group
Peking University
Beijing, China

Abstract

The in-domain performance of Binary Neural Networks (BNNs) has been significantly boosted by recent research efforts. The effect of heavy compression to the generalization capability of BNNs, however, has not been explored. This paper shows that binary compression degrades the generalization capability of BNNs, and addresses this issue by optimizing the distribution of BNN parameters and activations. A novel BNN training scheme is proposed to pursue a flat minimum for binary parameters through optimizing latent real-valued weights. Our method also optimizes the distributions of BNN activations to decrease the quantization errors caused by binarization. Extensive experiments on three domain generalization datasets reveal that, jointly optimizing BNN weights and activations substantially enhances the generalization capability, making our BNN achieve the best performance among its competitors. Our method also exhibits good compatibility to different network architectures, and performs well on general image classification datasets like CIFAR-100 and ImageNet.

1 Introduction

Binary Neural Networks (BNNs) apply binary compression to weights and activations of Floating point Neural Networks (FNNs) to save storage and computations. This compression strategy significantly accelerates the inference of FNNs, but leads to a fundamental challenge: what negative effects would such heavy compression bring to FNNs?

An easily observed effect is the degraded performance of BNNs. Thanks to recent efforts on binary network architectures and training strategies [26, 27], the in-domain performance of BNNs has been significantly improved. For instance, trained and tested on the large-scale ImageNet [6], BNNs have achieved similar image classification accuracy with FNNs [39]. However, the effects of binary compression to the out-of-domain performance of BNNs has not been explored. We compare the out-of-domain and in-domain performance of BNNs following setting from a recent work [3] on PACS [21]. Results show that, BNN achieves comparable performance with FNNs when training and testing datasets are in the same domain, *i.e.*, 93.6% *VS.* 93.7%. The performance gaps between BNN and FNN are substantially

larger on out-of-domain test sets 69.2% *VS.* 79.0%. This could be attributed to the degraded modeling capability for visual cues of BNNs, which makes them easier to get over-fitted to training domain.

As Domain Generalization (DG) capability is important for computer vision tasks, this paper aims to improve the DG capability of BNNs through enhancing their robustness to domain shifts. Binary weights and activations limit the capabilities of modeling visual cues. As proved in SWAD [3], seeking flat minima for CNN weights leads to better robustness against the domain shift [12, 13, 18, 19]. Inspired by previous DG methods, the robustness to domain shift can be optimized by finding flat minima in real-valued weights [3]. Also, BNN binarizes real-valued activations before convolution. Smaller quantization error is helpful to preserve the DG capability of real-valued activations. We therefore optimize distributions of both weights and activations of BNNs.

There exist many methods [3] seeking flat minima for real-valued CNNs. However, they can not be directly adopted to BNNs because of discrete values ± 1 in BNN weights. To address this issue, we optimize a real-valued FNN to pursue a flat minimum, meanwhile optimize weights in FNN to make each of them get close to +1 or -1. This FNN is hence binarized as the BNN. This strategy effectively decreases quantization errors caused by binarization, as well as boosts the DG capability of BNNs. For BNN activations, we optimize their distributions by reducing the quantization error and producing even-distribution activations at different layers using an activation regularization loss.

Experiments are conducted using different network architectures on three widely used DG datasets. Our method achieves substantially better performance than recent BNN methods like Bi-Real Net [24] and ReActNet [25] on PACS. We also applied recent DG methods to BNNs. Experiments show, our method also outperforms recent DG methods such as SWAD [3] and MixStyle[42]. Those experiments clearly reveal the validity of proposed training strategies in enhancing DG capability of BNNs. To the best of our knowledge, this is an early work studying the out-of-domain performance of BNNs. It reveals that, binary compression is harmful to DG capability of neural networks. Besides, the DG capability of BNNs can be effectively recovered through jointly optimizing a flat minimum in binary parameters and applying an activation regularization loss on the BNN activations.

2 Related Work

This work is related to domain generalization and binary neural network optimization. This section briefly reviews recent works in those two categories.

Domain generalization (DG) aims to enhance the out-of-domain performance on source data typically composed of multiple related but distinct domains. Most DG methods [15, 22, 28, 40] learn domain-invariant representations through aligning features of different domains. For instance, MDA [15] learns a domain-invariant feature transformation to learn distinctive features. Recently, meta-learning has been applied on DG [8, 9, 41]. Data Augmentation is another category for learning domain-invariant models [29, 35, 42]. It regularizes the model to avoid overfitting. MixStyle [42] increases source domains diversity by mixing styles of source domains to boost the generalizability of the trained model. Some DG methods leverage ensemble learning using multiple copies of the same model to boost the performance of a single model [3, 10, 44]. By averaging the converged models, SWAD [3] can find a flat minimum and suffers less from overfitting.

Binary Neural Networks (BNNs) enjoy high efficiency but suffer from degraded per-

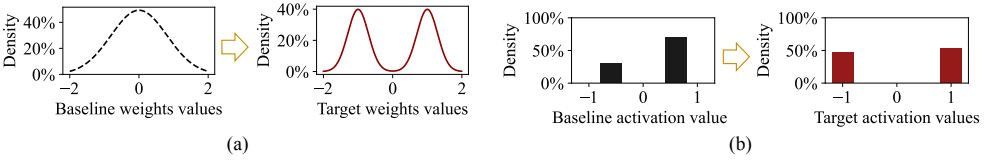

Figure 1: Illustration of baseline and expected distributions for weights and activations.

formance. Many works enhance the performance of BNNs by designing better network structures [23, 24, 25, 27, 45]. For example, Bi-Real Net [24] and ReActNet [25] add real-valued shortcuts with marginal computation overheads. Some other works focus on better optimization for BNNs [2, 7, 26, 30, 31]. For example, XNOR-Net [31] introduces binary convolutional kernels with scalars. XNOR-Net++ [2] further proposes to learn scalars with three parameters corresponding to three dimensions of kernels. Liu *et al.* [26] analyze the Adam training strategies and propose better solutions for optimizing BNNs. IR-Net [30] proposes Libra-PB to minimize both quantization error and information loss simultaneously. [7] proposes activation regularization loss to improve BNN training.

This work aims to boost the DG capability of BNNs, thus differs with previous BNN works in both motivation and methodologies. We optimize both distributions of weights and activations in BNNs to achieve better DG. BONN [14] proposes to reduce quantization error between latent weights and binarized weights. IR-Net [30] gets informative weights and activations by balancing the distribution. Different from BONN and IR-Net, our method optimizes the distribution of weights to find a flat minimum. Previous works [33] and [37] also train latent weights as a separated real-valued network. Differently, we optimize the real-valued network with parallel training to chase a flat minimum. Our experiments compare with recent BNNs and DG methods. Experiment results show our method outperforms existing BNNs and DG methods.

3 Proposed Methods

3.1 Overview

Given a BNN with N-layers, we denote binary weights and real-valued latent weights as $W = \{\omega_n\}_{n=1}^{N}$ and $\hat{W} = \{\hat{\omega}_n\}_{n=1}^{N}$, respectively. $\mathbf{D} = \{(x_m, y_m, d_m)\}_{m=1}^{M}$ denotes a source domain dataset with M images in K domains, where x_m, y_m and $d_m \in \{1 : K\}$ denote the image, label and domain index respectively. Target domain dataset with J images are denoted as $\mathbf{T} = \{(x_j, y_j, d_j)\}_{j=1}^{J}$, where $d_j = K+1$. Our goal is to use data from $\mathbf{D}$ to train a compact BNN model which can generalize well to an unseen domain $\mathbf{T}$.

To enhance the domain generalization (DG) capability of BNNs, we optimize distribution of both its weights and activations. A flat minimum in parameter space can absorb disturbance brought by domain shift and increase the DG capability of a network. We optimize distributions of real-valued weights to find a flat minimum, meanwhile learn real-valued weights close to +1/-1 as shown in Fig. 1 (a) to preserve the flat minimum in binarized weights. Also, BNN activations may suffer from quantization errors during binarization. Binarized activations should also be evenly distributed between -1 and +1 to enhance their capability in encoding meaningful cues. We hence also propose a regularization loss on BNN activations to pursue target distribution illustrated in Fig. 1 (b). The overall loss function $\mathcal{L}$

for network training can be denoted as

$$\mathcal{L} = \mathcal{L}^{\mathrm{B}} + \beta\mathcal{L}^{\mathrm{F}} + \alpha\mathcal{L}^{G} + \gamma\mathcal{L}^{A}, \tag{1}$$

where $\mathcal{L}^{\mathrm{B}}$ denotes the task specific loss calculated using binarized weights W. $\mathcal{L}^{\mathrm{F}}$ denotes the task specific loss calculated using real-valued weights with disturbance to find a flat minimum. $\mathcal{L}^{G}$ is the gap loss to measure the distance between binarized weights W and real-valued weights $\hat{W}$. It is minimized to preserve the flat minima of $\hat{W}$ in binary weights. $\mathcal{L}^{A}$ denotes the activation regularization loss to optimize the distribution of activations. α, β, γ are the weights for loss functions. The following parts present details of training procedure.

3.2 Baseline Method for BNN Optimization

We first revisit the forward propagation and training of BNNs with $\mathcal{L}^{\mathrm{B}}$. Both activations and weights at the n-th layer are first binarized by the sign function $\mathbb{S}(\cdot)$

$$O_n^{\mathrm{B}} = \mathbb{S}(\hat{O}_n^{\mathrm{B}}) = \begin{cases} -1, \hat{O}_n^{\mathrm{B}} < 0, \\ +1, \hat{O}_n^{\mathrm{B}} \geq 0, \end{cases} \quad \omega_n = \frac{\|\hat{\omega}_n\|_1}{s_n}\mathbb{S}(\hat{\omega}_n) = \begin{cases} -\frac{\|\hat{\omega}_n\|_1}{s_n}, \hat{\omega}_n < 0, \\ +\frac{\|\hat{\omega}_n\|_1}{s_n}, \hat{\omega}_n \geq 0, \end{cases} \tag{2}$$

where O_n^{B} is the binarized activations from $\hat{O}_n^{\mathrm{B}}$ and the superscript ^{B} denotes they are calculated with binary weights. $\hat{\omega}_n$ is binarized and normalized by $\frac{\|\hat{\omega}_n\|_1}{s_n}$ as ω_n [31]. s_n is the size of the weight tenser in the n-th layer. The n-th layer hence convolves O_n^{B} with ω_n and outputs activation $\hat{O}_{n+1}^{\mathrm{B}}$ after a BN layer, *i.e.*,

$$\hat{O}_{n+1}^{\mathrm{B}} = \mathrm{bn}(\mathrm{Conv}(\omega_n, O_n^{\mathrm{B}})). \tag{3}$$

The derivative of the sign function $\mathbb{S}(\cdot)$ is zero for most inputs, making it incompatible with backward propagation. To end-to-end train the BNN, real-valued weights $\hat{\omega}_n$ are hence stored as latent weights for gradients accumulation. This training strategy is commonly known as Straight-Through Estimator (STE) [17, 24, 27]. The network is trained with a task specific loss $\mathcal{L}^{\mathrm{B}}$ calculated on outputs computed by binarized weights, *i.e.*,

$$\mathcal{L}^{\mathrm{B}} = \frac{1}{M}\sum_{m=1}^{M} \mathrm{U}(\mathrm{B}(W; x_m, d_m), y_m), \tag{4}$$

where $\mathrm{B}(\cdot;\cdot,\cdot)$ computes the BNN prediction with binary weights. $\mathrm{U}(\cdot,\cdot)$ is the loss function computed with ground truth y. The above training objective is commonly used in baseline methods. Following parts introduce our strategies to enhance the DG capability of BNN.

3.3 Flat Minima Optimization on BNN Weights

Many works apply random disturbances to CNN weights to seek flat minima [3]. It is difficult to directly optimize the flat minima on BNNs in this way, because BNNs use binarized weights for forward propagation. Therefore, small random disturbance Δ can not bring sign flips for binarized weights W. Large disturbances Δ can not lead to a close neighborhood for BNN weights. Instead of directly learning a flat minimum for W, we find the flat minima for real-valued weights $\hat{W}$, meanwhile minimize the differences between W and $\hat{W}$.

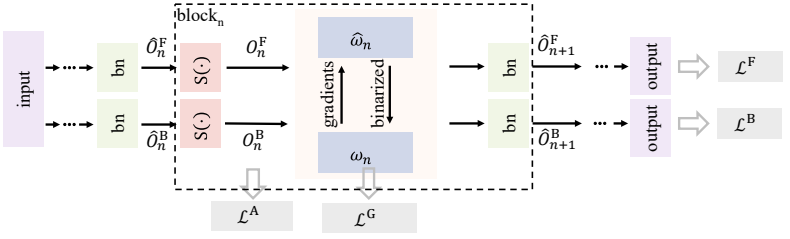

Figure 2: Illustration of the pipeline for our Binary Neural Network training.

We introduce a FNN having the identical structure with BNN. This FNN uses real-valued weights and binarized activations for forward propagation. This network can be trained by $\mathcal{L}^{\mathrm{F}}$, a task specific loss over multiple source domains,

$$\mathcal{L}^{\mathrm{F}}(\hat{W}) = \frac{1}{M}\sum_{m=1}^{M} \mathrm{U}(\mathrm{F}(\hat{W}; x_m, d_m), y_m), \tag{5}$$

where $\mathrm{F}(\cdot;\cdot,\cdot)$ computes the FNN prediction. A solution to a flat minimum can be found by solving $\arg\min_{\hat{W}} \mathcal{L}_{\mu}^{\mathrm{F}}(\hat{W})$ within a neighborhoods bounded by μ, *i.e.*,

$$\arg\min_{\hat{W}} \mathcal{L}_{\mu}^{\mathrm{F}}(\hat{W}) = \arg\min_{\hat{W}} \max_{\|\Delta\|_2 \le \mu} \mathcal{L}^{\mathrm{F}}(\hat{W}+\Delta), \tag{6}$$

where μ defines a neighborhood of $\hat{W}$. If $\mathcal{L}^{\mathrm{F}}(\hat{W}+\Delta)$ keeps small loss within the neighborhood, there exists the local optimum within μ-ball [3], which is regarded as a flat minimum.

The introduced FNN updates real-valued weights $\hat{W}$ for gradients accumulation and to find a flat minimum. As shown in Fig. 2, the n-th block of the FNN takes $\hat{O}_n^{\mathrm{F}}$ as inputs, where the superscript $^{\mathrm{F}}$ denotes the activation is convolved by real-valued weights. $\hat{O}_n^{\mathrm{F}}$ is binarized as O_n^{F}, then is convolved with real-valued weight $\hat{\omega}_n$ plus random Gaussian disturbance Δ, where the mean and variance of the disturbances are 0 and $\|\hat{\omega}_n\|_1/2s_n$ respectively. The above computations output an activation $\hat{O}_{n+1}^{\mathrm{F}}$, *i.e.*,

$$\hat{O}_{n+1}^{\mathrm{F}} = \mathrm{bn}(\mathrm{Conv}(\hat{\omega}_n + \Delta, O_n^{\mathrm{F}})), \tag{7}$$

where, $\mu = \|\hat{\omega}_n\|_1/2s_n$, we apply Δ as a disturbance for $\hat{W}$. The network is end-to-end trained with task specific loss $\mathcal{L}^{\mathrm{F}}$ calculated on FNN outputs in similar way of Eq. (5).

After training convergence, the random disturbances would find a flat minimum for $\hat{W}$. As $\hat{W}$ is binarized as BNN weights, the distance between W and $\hat{W}$ should be minimized to preserve the flat minima. Therefore, we propose $\mathcal{L}^{G}$ as the measurement of the differences between W and $\hat{W}$.

$$\mathcal{L}^{G} = \sum_{n=1}^{N} \|(\hat{\omega}_n - \omega_n)\|_2. \tag{8}$$

Eq. 8 minimizes the differences between binarized and real-valued weights. It hence restricts each of real-valued weights close to +1 or -1. This loss effectively decreases the quantization errors caused by parameter binarization, hence preserves the flat minima learned for $\hat{W}$ and makes the training stage more stable.

3.4 Regularization on BNN Activations

BNN binarizes activations before convolution, which leads to quantization errors and considerable information loss. To enhance the DG capability, BNN activations should be optimized to decrease quantization error. Also, binary activations should be evenly distributed between +1 and -1 to enhance their capability in encoding meaningful cues, as in Fig. 1 (b).

We denote a batch of real-valued activations of the n-th layer as:

$$\hat{O}_n = [\hat{o}_n^1, \hat{o}_n^2, ..., \hat{o}_n^R], \tag{9}$$

where R is the batch size. $\hat{o}_n^r(p)$ is the p-th location of $\hat{o}_n^r$, $p = 1 : P_n$. Regularization loss for the n-th layer $\mathcal{L}_n^A$ can be defined as (we omit $_n$ in Sec 3.4 to simplify the description),

$$\mathcal{L}^A = \mathbf{E} + \mathbf{G}, \tag{10}$$

where $\mathbf{E}$ and $\mathbf{G}$ denote quantization error loss and even-distribution loss respectively.

Since the gradient approximation of STE clips the real-valued activations to $[-1, +1]$, we can decrease the quantization error of each $\hat{o}_n^r(p)$ by just pushing it away from 0. So we define the quantization error loss $\mathbf{E}$ as,

$$\mathbf{E} = \frac{1}{P \cdot R} \sum_{p=1}^{P} \sum_{r=1}^{R} -(\hat{o}^r(p))^2, \tag{11}$$

where an $\hat{o}_n^r(p)$ close to 0 leads to larger loss.

The even-distribution loss is applied to encode more meaningful cues at each $\hat{o}_n^r(p)$. For instance, an $\hat{o}_n^r(p)$ producing similar numbers of $+1$ and -1 on a large dataset denotes its high entropy and good information encoding capability. To implement the even-distribution loss, we compute the mean value of $\hat{o}_n^r(p)$ at each location p over the training batch. Intuitively, this mean value close to 0 leads to a small loss value of $\mathbf{G}$, i.e.,

$$\mathbf{G} = \frac{1}{P} \sum_{p=1}^{P} (\frac{1}{R} \sum_{r=1}^{R} \hat{o}^r(p))^2. \tag{12}$$

Therefore, the regularization loss applied to the n-th layer $\mathcal{L}_n^A$ can be formulated as

$$\mathcal{L}^A = \frac{1}{P} \sum_{p=1}^{P} (\frac{1}{R} \sum_{r=1}^{R} -(\hat{o}^r(p))^2 + (\frac{1}{R} \sum_{r=1}^{R} \hat{o}^r(p))^2) = -\frac{1}{P} \sum_{p=1}^{P} \sigma^2(p), \tag{13}$$

where $\sigma(p)$ computes variance at the p-th location over the training batch. Notice that, minimizing the regularization loss equals to maximizing the variance at each location of the activation over the training batch.

4 Experiments

4.1 Datasets and Implementation Details

We follow Domain Generalization (DG) works [43] to conduct experiments on three datasets PACS [21], VLCS [11], and OfficeHome[34], respectively. For PACS and VLCS, we follow the setting in [3, 16] to leave one domain as the target domain and regard the others as source

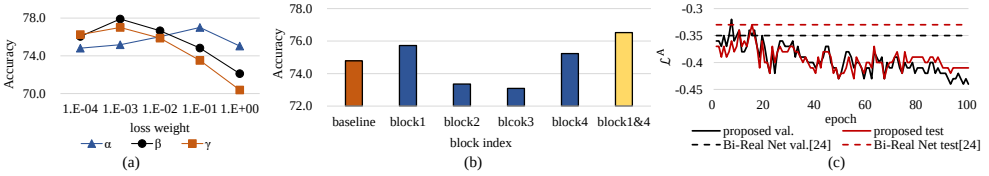

Figure 3: (a) Performance with different α, β and γ. (b) performance with $\mathcal{L}^A$ applied on different blocks. (c) Value of $\mathcal{L}^A$ on validation and test sets. Results are reported on PACS.

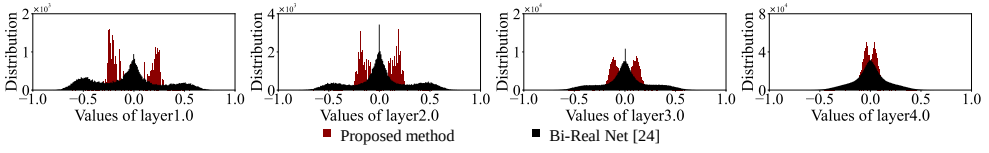

Figure 4: Illustration of final weight distribution of models from baseline and our method. More visualizations can be found in Supp. A.1.

domain. OfficeHome does not provide the validation set. We choose the model trained by the last epoch for evaluation. Besides DG datasets, we further test our method on traditional classification datasets including CIFAR-100 [20] and ImageNet [6].

We apply the proposed method to different BNNs including Bi-Real Net [24] and ReAct-Net [25], respectively. We follow the training setting in a BNN work [26] and use pre-trained model from ImageNet. Notice that, existing works [24, 25, 39] do not binarize the first convolutional layer, the last FC layer, nor the 1×1 convolution layers in network. We also follow this setting. For domain generalization tasks, we first initialize the BNN weights pre-trained on ImageNet [6] following [16]. For both network structures, we use the pre-trained model from [25]. After that, we fine-tune the model with source datasets for 100 epochs. The initial learning rate is set as $1e-3$ and decreases by 0.1 at epoch 80. Following [26], we use Adam optimizer with weight decay set as $2e-6$. For CIFAR-100 and ImageNet, we follow recent works [25] to use Bi-Real Net and the two-step training strategy.

4.2 Ablation Study

Analysis on loss weight α, β and γ: The impacts of loss weights α, β and γ on the BNN performance are summarized in Fig. 3 (a), where each curve is plotted by varying one parameter and fixing the other two. Large α helps to minimize the differences between real-valued and binarized weights, hence boosts the performance. Too large α makes the training convergence difficult and degrades the performance. A reasonably large β makes $\mathcal{L}^{\mathrm{F}}$ more important for BNN training, hence helps to seek the flat minima. Setting a larger γ also gets the best performance. We fix α, β, γ as 0.1, 0.001, and 0.001, respectively on other datasets, where they generalize and perform well as shown in subsequent experiments.

Analysis on regularization loss: Each binary convolutional block outputs a real-valued activation. Therefore, the activation regularization loss $\mathcal{L}^A$ can be applied on different blocks. Our experiments show that, adding $\mathcal{L}^A$ at different blocks leads to different performance. We conduct experiments on Bi-Real Net [24] consisting of 4 blocks. The first and the last layers

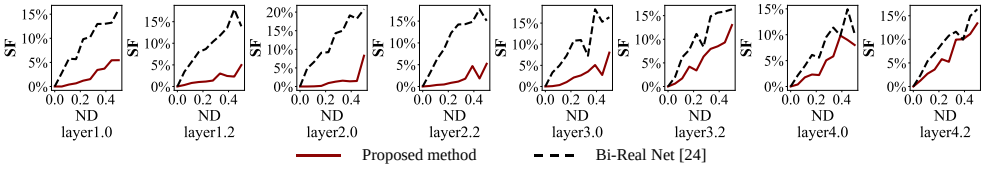

Figure 5: Illustration of the sign flip rates (**SF**) of weights with turbulence of different Noise Degree (ND) from 0 to 0.5. Smaller **SF** indicates the weights are more stable with turbulence.

Methods	A	C	P	S	Mean
ResNet-18 [16]†	83.4	80.3	96.0	80.9	85.1
IR-Net [30]	70.4	72.4	87.8	73.5	76.0
ReCU [38]	70.5	73.1	87.0	71.2	75.45
Bi-Real Net [24]	69.2	72.6	86.7	70.6	74.8
+RSC [16]	65.1	71.5	85.2	67.2	72.3
+SWAD [3]	67.3	72.9	87.0	74.0	75.3
+MixStyle [42]	69.5	72.3	87.0	70.9	74.9
+MIRO [4]	69.9	72.9	87.3	71.2	75.3
$+\mathcal{L}^A$	69.6	72.9	88.9	74.7	76.5
$+\mathcal{L}^G+\mathcal{L}^F$	72.0	73.5	88.7	74.9	77.3
ours	**72.4**	**73.7**	**89.8**	**75.5**	**77.8**
ReActNet [25]	66.4	68.5	85.6	75.7	74.0
$+\mathcal{L}^G+\mathcal{L}^F$	**72.3**	72.9	**90.4**	74.2	77.5
ours	72.2	**73.9**	89.3	**75.7**	**77.8**

Table 1: Comparison on PACS, with P (Photo), A (Art-painting), C (Cartoon) and S (Sketch).

are preserved as real-valued in many BNNs methods [24, 25, 27] because they are more important in ensuring a high performance. We apply $\mathcal{L}^A$ to different blocks and summarize the results in Fig. 3 (b). It is clear that, applying $\mathcal{L}^A$ to the first and the last binarized block brings steady performance improvement over the baseline. Applying $\mathcal{L}^A$ on both block1&4 further boosts the performance. In following experiments, we apply $\mathcal{L}^A$ on the first and the last binarized block for both Bi-Real Net and ReActNet. To analyze the effectiveness of $\mathcal{L}^A$, we further show its value during training in Fig. 3 (c), where the loss value is decreased on both validation and test sets. This also indicates that the variance of activation is enlarged by $\mathcal{L}^A$, which enables it to encode more cues and boost the performance.

Analysis on weight distribution: Real-valued latent weights are optimized by $\mathcal{L}^F$ to find a flat minimum. This experiment analyzes the weight distributions learned by our method. As shown in Fig. 4, the distribution of weights in Bi-Real Net [24] trained with [26] shows a peak around 0. Distribution of latent weights trained by our method shows two peaks, respectively. It is easy to infer that, latent weights learned by [26] are sensitive to disturbance, e.g., a small disturbance easily changes the binarized weights. Weights learned by our method are more robust to disturbance, and produce smaller quantization errors after binarization. We further quantify BNN stability by measuring the sign flip rate, which also shows that our method is more robust to noise turbulence as shown in Fig. 5. More computation details and visualizations are provided in Supp. A.1.

Methods	R	P	C	A	Mean
ResNet-18 [32]†	73.2	71.8	44.2	58.7	62.0
Bi-Real Net [24]	64.9	65.6	41.3	43.2	53.8
$+\mathcal{L}^G + \mathcal{L}^F$	64.8	65.6	43.1	43.7	54.3
ours	**66.0**	**66.1**	**43.3**	**44.6**	**55.0**
ReActNet [25]	63.0	63.8	44.6	40.6	53.0
$+\mathcal{L}^G + \mathcal{L}^F$	67.0	**67.9**	45.5	46.6	56.7
ours	**67.1**	67.6	**45.6**	**47.9**	**57.0**

Table 2: Comparison on OfficeHome with A(Art), R(Real), P(Product) and C(Clipart).

Methods	S	P	L	C	Mean
ResNet-18†	67.0	69.7	60.6	94.6	73.0
Bi-Real Net [24]	59.6	64.7	59.7	92.3	69.1
$+\mathcal{L}^G + \mathcal{L}^F$	61.7	67.1	60.7	95.8	71.3
ours	**62.1**	**67.8**	**62.4**	**96.2**	**72.1**
ReActNet [25]	61.4	60.4	61.2	93.2	69.1
$+\mathcal{L}^G + \mathcal{L}^F$	**62.6**	**67.2**	62.2	95.3	71.8
ours	62.1	66.9	**62.9**	**96.0**	**72.0**

Table 3: Comparison on VLCS with S (Sun), P (Pascal), L (LabelMe) and C (Caltech).

Methods	A	C	P	S	Mean
Bi-Real Net w/o $\mathcal{L}^A$ [24]	48.7	60.4	72.0	59.2	60.1
Bi-Real Net w/ $\mathcal{L}^A$	52.8	63.7	76.1	61.8	63.6

Table 4: Comparison on PACS with Bi-Real Net without ImageNet pre-trained.

Methods	A	C	P	S	Mean
ReActNet [25]	58.9	67.6	85.5	57.4	67.4
ours	60.4	69.2	88.4	63.9	70.5

Table 5: Comparison on PACS with ReActNet with backbone fixed.

4.3 Comparison With Recent Works

Comparison on DG datasets: This part first compares with existing BNNs and DG methods on three commonly used DG datasets. Table 1 summarizes the comparison on PACS, where the performance of the real-valued ResNet-18 is also reported [16]. Currently, there is no BNN method working on DG. We hence apply different DG methods designed for real-valued networks [3, 16, 42] on Bi-Real Net. Results show that these DG methods do not boost the DG performance of Bi-Real Net. Applying $\mathcal{L}^A$ or $\mathcal{L}^G + \mathcal{L}^F$ to Bi-Real Net brings performance gains as shown in Table 1. Our method combines those losses and achieves the best performance, *e.g.*, outperforms Bi-Real Net by 3.0% in accuracy. Our method also outperforms other BNNs methods [30, 38], and also boosts the performance of ReActNet.

To further evaluate the effectiveness of our method, we conduct experiments on datasets OfficeHome and VLCS and summarize the comparison in Table 2 and 3 respectively. On those datasets, we can get similar conclusion as shown in Table 1, i.e., our method consistently boosts the DG performance of BNNs like Bi-Real Net and ReActNet. Experiment results in Table 1, 2 and 3 also clearly show the validity of our proposed losses.

We also test the effectiveness of our method with two different training setups. One setup does not use ImageNet for pre-training, hence makes the BNN get easily overfitted to the training set. As shown in Table 4, $\mathcal{L}^A$ brings a more substantial performance enhancement in this training setup. The other setup fixes the backbone pre-trained on ImageNet and fine-tunes the FC layer on PACS source dataset. As shown in Table 5, our method outperforms the ReActNet by 3.1%. We hence could conclude that, our method is effective in boosting the DG of BNNs for both setups with or without pre-training.

Comparison on CIFAR-100 and ImageNet: We also evaluate our method on CIFAR-100 [20] and ImageNet [6]. In Table 6, our method achieves the accuracy of 70.53% on CIFAR-100 [20], outperforming recent methods [5, 26]. In Table 7, our method achieves

Method	Top-1 Acc (%)
Bi-Real Net [5]	68.78
+HOW [26]	68.90
$+\mathcal{L}^A$	70.15
$+\mathcal{L}^G + \mathcal{L}^F$	70.40
ours	**70.53**

Table 6: Comparison on CIFAR-100 with Bi-Real Net following [5].

Method	GFLOPs	Top-1 Acc (%)
CI-BCNN [36]	0.154	56.7
R2B Net [27]	0.165	65.4
MeliusNet29 [1]	0.214	65.8
ReActNet [25]	0.087	69.4
ours	0.087	68.9

Table 7: Comparison on ImageNet with recent SOTA BCNN methods.

comparable performance with ReActNet [25] and outperforms other competitors in aspects of both accuracy and efficiency.

5 Conclusion

This paper shows that BNNs presents degraded DG capability compared with FNNs, and addresses this issue by optimizing the distribution of BNN parameters and activations. We optimize real-valued latent weights to pursue a flat minimum, meanwhile minimize differences between real-valued and binarized weights. Our method also optimizes the distributions of BNN activations to decrease the quantization errors caused by binarization. Extensive experiments on five datasets reveal the promising performance of the proposed method.

Acknowledgement: This work is supported in part by The National Key Research and Development Program of China under Grant No. 2018YFE0118400, in part by Natural Science Foundation of China under Grant No. U20B2052, 61936011.

References

[1] Joseph Bethge, Christian Bartz, Haojin Yang, Ying Chen, and Christoph Meinel. Meliusnet: Can binary neural networks achieve mobilenet-level accuracy? *arXiv preprint arXiv:2001.05936*, 2020.

[2] Adrian Bulat and Georgios Tzimiropoulos. Xnor-net++: Improved binary neural networks. In *BMVC*, 2019.

[3] Junbum Cha, Sanghyuk Chun, Kyungjae Lee, Han-Cheol Cho, Seunghyun Park, Yunsung Lee, and Sungrae Park. Swad: Domain generalization by seeking flat minima. *arXiv preprint arXiv:2102.08604*, 2021.

[4] Junbum Cha, Kyungjae Lee, Sungrae Park, and Sanghyuk Chun. Domain generalization by mutual-information regularization with pre-trained models. *European Conference on Computer Vision (ECCV)*, 2022.

[5] Tianlong Chen, Zhenyu Zhang, Xu Ouyang, Zechun Liu, Zhiqiang Shen, and Zhangyang Wang. "bnn - bn = ?": Training binary neural networks without batch normalization. In *CVPR Workshops*, 2021.

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

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

[8] Qi Dou, Daniel Coelho de Castro, Konstantinos Kamnitsas, and Ben Glocker. Domain generalization via model-agnostic learning of semantic features. *NeurIPS*, 2019.

[9] Yingjun Du, Jun Xu, Huan Xiong, Qiang Qiu, Xiantong Zhen, Cees GM Snoek, and Ling Shao. Learning to learn with variational information bottleneck for domain generalization. In *ECCV*, 2020.

[10] Antonio DInnocente and Barbara Caputo. Domain generalization with domain-specific aggregation modules. In *German Conference on Pattern Recognition*, 2018.

[11] Chen Fang, Ye Xu, and Daniel N Rockmore. Unbiased metric learning: On the utilization of multiple datasets and web images for softening bias. In *ICCV*, 2013.

[12] Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. Sharpness-aware minimization for efficiently improving generalization. *arXiv*, 2020.

[13] Timur Garipov, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry P Vetrov, and Andrew G Wilson. Loss surfaces, mode connectivity, and fast ensembling of dnns. *NeurIPS*, 2018.

[14] Jiaxin Gu, Junhe Zhao, Xiaolong Jiang, Baochang Zhang, Liu Jianzhuang, Guodong Guo, and Rongrong Ji. Bayesian optimized 1-bit cnns. In *ICCV*, 2019.

[15] Shoubo Hu, Kun Zhang, Zhitang Chen, and Laiwan Chan. Domain generalization via multidomain discriminant analysis. In *Uncertainty in Artificial Intelligence*, 2020.

[16] Zeyi Huang, Haohan Wang, Eric P Xing, and Dong Huang. Self-challenging improves cross-domain generalization. In *ECCV*, 2020.

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

[18] Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. *arXiv*, 2018.

[19] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv*, 2016.

[20] Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009.

[21] Da Li, Yongxin Yang, Yi-Zhe Song, and Timothy M Hospedales. Deeper, broader and artier domain generalization. In *ICCV*, 2017.

[22] Haoliang Li, YuFei Wang, Renjie Wan, Shiqi Wang, Tie-Qiang Li, and Alex C Kot. Domain generalization for medical imaging classification with linear-dependency regularization. *arXiv preprint arXiv:2009.12829*, 2020.

[23] Chunlei Liu, Wenrui Ding, Xin Xia, Baochang Zhang, Jiaxin Gu, Jianzhuang Liu, Rongrong Ji, and David Doermann. Circulant binary convolutional networks: Enhancing the performance of 1-bit dcnns with circulant back propagation. In *CVPR*, 2019.

[24] Zechun Liu, Baoyuan Wu, Wenhan Luo, Xin Yang, Wei Liu, and Kwang-Ting Cheng. Bi-real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm. In *ECCV*, 2018.

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

[26] Zechun Liu, Zhiqiang Shen, Shichao Li, Koen Helwegen, Dong Huang, and Kwang-Ting Cheng. How do adam and training strategies help bnns optimization? *arXiv preprint arXiv:2106.11309*, 2021.

[27] Brais Martinez, Jing Yang, Adrian Bulat, and Georgios Tzimiropoulos. Training binary neural networks with real-to-binary convolutions. In *ICLR*, 2019.

[28] Toshihiko Matsuura and Tatsuya Harada. Domain generalization using a mixture of multiple latent domains. In *AAAI*, 2020.

[29] Fengchun Qiao, Long Zhao, and Xi Peng. Learning to learn single domain generalization. In *CVPR*, 2020.

[30] 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.

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

[32] Seonguk Seo, Yumin Suh, Dongwan Kim, Geeho Kim, Jongwoo Han, and Bohyung Han. Learning to optimize domain specific normalization for domain generalization. In *ECCV*, 2020.

[33] Yuzhang Shang, Dan Xu, Ziliang Zong, Liqiang Nie, and Yan Yan. Contrastive mutual information maximization for binary neural networks. 2021.

[34] Hemanth Venkateswara, Jose Eusebio, Shayok Chakraborty, and Sethuraman Panchanathan. Deep hashing network for unsupervised domain adaptation. In *CVPR*, 2017.

[35] Riccardo Volpi and Vittorio Murino. Addressing model vulnerability to distributional shifts over image transformation sets. In *ICCV*, 2019.

[36] Ziwei Wang, Jiwen Lu, Chenxin Tao, Jie Zhou, and Qi Tian. Learning channel-wise interactions for binary convolutional neural networks. In *CVPR*, 2019.

[37] Weixiang Xu, Qiang Chen, Xiangyu He, Peisong Wang, and Jian Cheng. Improving binary neural networks through fully utilizing latent weights. *arXiv preprint arXiv:2110.05850*, 2021.

[38] Zihan Xu, Mingbao Lin, Jianzhuang Liu, Jie Chen, Ling Shao, Yue Gao, Yonghong Tian, and Rongrong Ji. Recu: Reviving the dead weights in binary neural networks. In *ICCV*, 2021.

[39] Jianming Ye, Jingdong Wang, and Shiliang Zhang. Distillation-guided residual learning for binary convolutional neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[40] Shanshan Zhao, Mingming Gong, Tongliang Liu, Huan Fu, and Dacheng Tao. Domain generalization via entropy regularization. *NeurIPS*, 2020.

[41] Yuyang Zhao, Zhun Zhong, Fengxiang Yang, Zhiming Luo, Yaojin Lin, Shaozi Li, and Nicu Sebe. Learning to generalize unseen domains via memory-based multi-source meta-learning for person re-identification. In *CVPR*, 2021.

[42] Kaiyang Zhou, Yongxin Yang, Yu Qiao, and Tao Xiang. Domain generalization with mixstyle. In *ICLR*, 2020.

[43] Kaiyang Zhou, Ziwei Liu, Yu Qiao, Tao Xiang, and Chen Change Loy. Domain generalization: A survey. *arXiv preprint arXiv:2103.02503*, 2021.

[44] Kaiyang Zhou, Yongxin Yang, Yu Qiao, and Tao Xiang. Domain adaptive ensemble learning. *IEEE Transactions on Image Processing*, 2021.

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