Binary domain generalization for sparsifying binary neural networks*

Riccardo Schiavone[1,2] [0000−0002−5089−7499], Francesco Galati[2] [0000−0001−6317−6298], and Maria A. Zuluaga[2] [0000−0002−1147−766X]

[1] Department of Electronics and Telecommunications, Politecnico di Torino, Italy
riccardo.schiavone@polito.it
[2] Data Science Department, EURECOM, Sophia Antipolis, France
{galati,zuluaga}@eurecom.fr

Abstract. Binary neural networks (BNNs) are an attractive solution for developing and deploying deep neural network (DNN)-based applications in resource constrained devices. Despite their success, BNNs still suffer from a fixed and limited compression factor that may be explained by the fact that existing pruning methods for full-precision DNNs cannot be directly applied to BNNs. In fact, weight pruning of BNNs leads to performance degradation, which suggests that the standard binarization domain of BNNs is not well adapted for the task. This work proposes a novel more general binary domain that extends the standard binary one that is more robust to pruning techniques, thus guaranteeing improved compression and avoiding severe performance losses. We demonstrate a closed-form solution for quantizing the weights of a full-precision network into the proposed binary domain. Finally, we show the flexibility of our method, which can be combined with other pruning strategies. Experiments over CIFAR-10 and CIFAR-100 demonstrate that the novel approach is able to generate efficient sparse networks with reduced memory usage and run-time latency, while maintaining performance.

Keywords: Binary neural networks · Deep neural networks · Pruning · Sparse representation.

1 Introduction

The increasing number of connected Internet-of-Things (IoT) devices, now surpassing the number of humans connected to the internet [6], has led to a sensors-rich world, capable of addressing real-time applications in multiple domains, where both accuracy and computational time are crucial [1]. Deep neural networks (DNNs) have the potential of enabling a myriad of new IoT applications, thanks to their ability to process large complex heterogeneous data and to extract patterns needed to take autonomous decisions with high reliability [20].

* FG and MAZ are supported by the French government, through the 3IA Côte d'Azur Investments in the Future project managed by the ANR (ANR-19-P3IA-0002)

However, DNNs are known for being resource-greedy, in terms of required computational power, memory, and energy consumption [4], whereas most IoT devices are characterized by limited resources. They usually have limited processing power, small storage capabilities, they are not GPU-enabled and they are powered with batteries of limited capacity, which are expected to last over 10 years without being replaced or recharged. These constraints represent an important bottleneck towards the deployment of DNNs in IoT applications [40].

A recent and notable example to enable the usage of DNNs in limited resource devices are binary neural networks (BNNs) [15]. BNNs use binary weights and activation functions that allow them to replace computationally expensive multiplication operations with low-cost bitwise operations during forward propagation. This results in faster inference and better compression rates, while maintaining an acceptable accuracy for complex learning tasks [10,25]. For instance, BNNs have achieved over 80% classification accuracy on ImageNet [10,31]. Despite the good results, BNNs have a fixed and limited compression factor compared to full-precision DNNs, which may be insufficient for certain size and power constraints of devices [22]. A way to further improve BNNs' compression capacity is through network pruning, which seeks to control a network's sparsity by removing parameters and shared connections [12]. Pruning BNNs, however, is a more challenging task than pruning full-precision neural networks and it is still a challenge with many open questions [38]. Current attempts [9,19,28,32,37,36,38] often rely on training procedures that require more training stages than standard BNNs, making learning more complex. Moreover, these methods fail in highly pruned scenarios, showing severe accuracy degradation over simple classification problems.

In this work, we introduce sparse binary neural network (SBNN), a more robust pruning strategy to achieve sparsity and improve the performance of BNNs. Our strategy relies on entropy to optimize the network to be largely skewed to one of the two possible weight values, i.e. having a very low entropy. Unlike BNNs that use symmetric values to represent the network's weights, we propose a more general binary domain that allows the weight values to adapt to the asymmetry present in the weights distribution. This enables the network to capture valuable information, achieve better representation, and, thus better generalization. The main contributions of our work can be summarized as follows: 1) We introduce a more general binary domain w.r.t. the one used by BNNs to quantize real-valued weights; 2) we derive a closed-form solution for binary values that minimizes quantization error when real-valued weights are mapped to the proposed domain; 3) we enable the regularization of the BNNs weights distribution by using entropy constraints; 4) we present efficient implementations of the proposed algorithm, which reduce the number of bitwise operations in the network proportionally to the entropy of the weight distribution; and 5) we demonstrate SBNN's competitiveness and flexibility through benchmark evaluations.

The remaining of this work is organized as follows. Section 2 discusses previous related works. The core of our contributions are described in Section 3. In Section 4, we study the properties of the proposed method and assess its perfor-

mance, in terms of accuracy and operation reduction at inference, through a set of experiments using, CIFAR-10, CIFAR-100 [18] and ImageNet [31] datasets. Finally, a discussion on the results and main conclusions are drawn in Section 5.

2 Related Work

We first provide an overview of BNNs. Next, we review sparsification through pruning [2,12,27,34] and quantization [11,16,39,41], the two network compression strategies this work relies on. A broad review covering further network compression and speed-up techniques can be found in [21].

Binary Neural Networks. BNNs [15] have gained attention in recent years due to their computational efficiency and improved compression. Subsequent works have extended [15] to improve its accuracy. For instance, [30] introduced a channel-wise scaling coefficient to decrease the quantization error. ABC-Net adopts multiple binary bases [23], and Bi-Real [26] recommends short residual connection to reduce the information loss and a smoother gradient for the signum function. Recently, ReActNet [25] generalized the traditional sign$(\cdot)$ and PReLU activation functions to extend binary network capabilities, achieving an accuracy close to full-precision ResNet-18 [13] and MobileNet V1 [14] on ImageNet [31]. By adopting the RSign, the RPReLU along with an attention formulation Guo et al. [10] surpassed the 80% accuracy mark on ImageNet. Although these works have been successful at increasing the performance of BNNs, few of them consider the compression aspect of BNNs.

Network Sparsification. The concept of sparsity has been well studied beyond quantized neural networks as it reduces a network's computational and storage requirements and it prevents overfitting. Methods to achieve sparsity either explicitly induce it during learning through regularization (e.g. L_0 [27] or L_1 [12] regularization), or do it incrementally by gradually augmenting small networks [2]; or by post hoc pruning [8,33,34].

BNNs pruning is particularly challenging because weights in the $\{\pm 1\}$ domain cannot be pruned based only on their magnitude. Existing methods include removing unimportant channels and filters from the network [9,28,37,38], but optimum metrics are still unclear; quantizing binary kernels to a smaller bit size than the kernel size [36]; or using the $\{0, \pm 1\}$ domains [19,32]. Although these works suggest that the standard $\{\pm 1\}$ binary domain has severe limitations regarding compression, BNNs using the $\{0, \pm 1\}$ domain have reported limited generalization capabilities [19,32]. In our work, we extend the traditional binary domain to a more general one, that can be efficiently implemented via sparse operations. Moreover, we address sparsity explicitly with entropy constraints, which can be formulated as magnitude pruning of the generic binary weight values mapping them in the $\{0, 1\}$ domain. In our proposed domain, BNNs are more robust to pruning strategies and show better generalization properties than other pruning techniques for the same sparsity levels.

Quantization. Network quantization allows the use of fixed-point arithmetic and a smaller bit-width to represent network parameters w.r.t the full-precision

counterpart. Representing the values using only a finite set requires a quantization function that maps the original elements to the finite set. The quantization can be done after training the model, using parameter sharing techniques [11], or during training by quantizing the weights in the forward pass, as ternary neural networks (TNNs) [17], BNNs [5] and other quantized networks do [16,39] . Our work builds upon the strategy of BNNs by introducing a novel quantization function that maps weights to a binary domain that is more general than the $\{\pm 1\}$ domain used in most state-of-the-art BNNs. This broader domain significantly reduces the distortion-rate curves of BNNs across various sparsity levels, enabling us to achieve greater compression.

3 Method

The proposed SBNN achieves network pruning via sparsification by introducing a novel quantization function that extends standard BNNs weight domain $\{\pm 1\}$ to a more generic binary domain $\{\alpha, \beta\}$ and a new penalization term in the objective loss controlling the entropy of the weight distribution and the sparsity of the network (Section 3.2). We derive in Section 3.3 the optimum SBNN's $\{\alpha, \beta\}$ values, i.e. the values that minimize the quantization loss when real-valued weights are quantized in the proposed domain. In Section 3.4, we use BNN's state-of-the-art training algorithms for SBNN training by adding the sparsity regularization term to the original BNN's objective loss. Section 3.5 describes the implementation details of the proposed SBNN to illustrate their speed-up gains w.r.t BNNs.

3.1 Preliminaries

The training of a full-precision DNN can be seen as a loss minimization problem:

$$\arg\min_{\widetilde{\mathbf{W}}} \mathcal{L}(y, \hat{y}) \tag{1}$$

where $\mathcal{L}(\cdot)$ is a loss function between the true labels y and the predicted values $\hat{y} = f(\mathbf{x}; \widetilde{\mathbf{W}})$, which are a function of the data input $\mathbf{x}$ and the network's full precision weights $\widetilde{\mathbf{W}} = \{\widetilde{\mathbf{w}}^\ell\}$, with $\widetilde{\mathbf{w}}^\ell \in \mathbb{R}^{N^\ell}$ the weights of the ℓ^{th} layer, and $N = \sum_\ell N^\ell$ the total number of weights in the DNN. We denote the i^{th} weight element of $\widetilde{\mathbf{w}}^\ell$ as $\widetilde{w}_i^\ell$.

A BNN [15] uses a modified signum function as quantization function that maps full precision weights $\widetilde{\mathbf{W}}$ and activations $\widetilde{\mathbf{a}}$ to the $\{\pm 1\}$ binary domain, enabling the use of low-cost bitwise operations in the forward propagation, i.e.

$$\overline{\mathbf{W}} = \text{sign}(\widetilde{\mathbf{W}}), \qquad \frac{\partial g(\widetilde{w}_i)}{\partial \widetilde{w}_i} = \begin{cases} \frac{\partial g(\widetilde{w}_i)}{\partial \overline{w}_i} & \text{, if } -1 \leq \widetilde{w}_i \leq 1 \\ 0 & \text{, otherwise,} \end{cases}$$

where sign(·) denotes the modified sign function over a vector, $g(\cdot)$ is a differentiable function, $\overline{\mathbf{W}}$ the network's weights in the $\{\pm 1\}$ binary domain, $\overline{w}_i$ a given weight in the binary domain, and $\widetilde{w_i}$ the associated full-precision weight.

3.2 Sparse Binary Neural Network (SBNN) Formulation

Given $\Omega^\ell = \{\alpha^\ell, \beta^\ell\}$ a general binary domain, with $\alpha^\ell, \beta^\ell \in \mathbb{R}$, and $\alpha^\ell < \beta^\ell$, let us define a SBNN, such that, for any given layer ℓ,

$$w_i^\ell \in \Omega^\ell \qquad \forall\ i, \tag{2}$$

with w_i^ℓ the i^{th} weight element of the weight vector, $\mathbf{w}^\ell$, and $\mathbf{w} = \left\{\mathbf{w}^\ell\right\}$ the set of weights for all the SBNN.

We denote S_{α^ℓ} and S_{β^ℓ} the indices of the weights with value α^ℓ, β^ℓ in $\mathbf{w}^\ell$

$$S_{\alpha^\ell} = \{i \,|\, 1 \leq i \leq N^\ell, w_i^\ell = \alpha^\ell\}, \qquad S_{\beta^\ell} = \{i \,|\, 1 \leq i \leq N^\ell, w_i^\ell = \beta^\ell\}.$$

Since $\alpha^\ell < \beta^\ell\ \forall\ \ell$, it is possible to estimate the number of weights taking the lower and upper values of the general binary domain over all the network:

$$L^\ell = |S_{\alpha^\ell}|, \qquad U^\ell = |S_{\beta^\ell}|, \qquad L = \sum_\ell L^\ell, \qquad U = \sum_\ell U^\ell, \tag{3}$$

with $L + U = N$, the total number of SBNN network weights. In the remaining of the manuscript, for simplicity and without loss of generality, please note that we drop the layer index ℓ from the weights notation.

To express the SBNN weights $\mathbf{w}$ in terms of binary $\{0, 1\}$ weights, we now define a a mapping function $r : \{0, 1\} \longrightarrow \{\alpha, \beta\}$ that allows to express $\mathbf{w}$:

$$w_i = r\left(w_{\{0,1\},i}\right) = \left(w_{\{0,1\},i} + \xi\right) \cdot \eta \tag{4}$$

with

$$\alpha = \xi \cdot \eta, \qquad \beta = (1 + \xi) \cdot \eta, \tag{5}$$

and $w_{\{0,1\},i} \in \{0, 1\}$, the i^{th} weight of a SBNN, when restricted to the binary set $\{0, 1\}$. Through these mapping, 0-valued weights are pruned from the network, the making SBNN sparse.

The bit-width of a SBNN is measured with the binary entropy $h()$ of the distribution of α-valued and β-valued weights,

$$h(p) = -p \log_2(p) - (1 - p) \log_2(1 - p) \qquad [\text{bits/weight}]\,, \tag{6}$$

with $p = U/N$. Achieving network compression using a smaller bit-width than that of standard BNN's weights (1 bit/weight) is equivalent to setting a constraint in the SBNN's entropy to be less or equal than a desired value h^*, i.e.

$$h(U/N) \leq h^*. \tag{7}$$

Given $h^{-1}()$ the inverse binary entropy function for $0 \leq p \leq 1/2$, it is straightforward to derive such constraint, $U \leq M$ where

$$M \triangleq N \cdot h^{-1}(h^*). \tag{8}$$

From Eq. (7) and (8), this implies that the constraint corresponds to restricting the maximum number of $1s$ in the network, and thus the sparsity of the network. Thus, the original full-precision DNN loss minimization problem (Eq. (1)) can be reformulated as:

$$\begin{aligned} \underset{\mathbf{w}_{\{0,1\}},\xi,\eta}{\arg\min} \quad & \mathcal{L}(y,\hat{y}) \\ \text{s.t.} \quad & \mathbf{w}_{\{0,1\}} \in \{0,1\}^N, \\ & U \leq M < N. \end{aligned} \tag{9}$$

The mixed optimization problem in Eq. (9) can be simplified by relaxing the sparsity constraint on U through the introduction of a non-negative function $g(\cdot)$, which penalizes the weights when $U > M$:

$$\begin{aligned} \underset{\mathbf{W}_{\{0,1\}},\xi,\eta}{\arg\min} \quad & \mathcal{L}(y,\hat{y}) + \lambda g(\mathbf{W}_{\{0,1\}}) \\ \text{s.t.} \quad & \mathbf{W}_{\{0,1\}} \in \{0,1\}^N \end{aligned} \tag{10}$$

and λ controls the influence of $g(\cdot)$. A simple, yet effective function $g(\mathbf{W}_{\{0,1\}})$ is the following one:

$$g\left(\mathbf{W}_{\{0,1\}}\right) = \text{ReLU}\left(U/N - \text{EC}\right), \tag{11}$$

where $\text{EC} = M/N$ represents the fraction of expected connections, which is the fraction of 1-valued weights in $\mathbf{W}_{\{0,1\}}$ over the total number of weights of $\mathbf{W}_{\{0,1\}}$.

Eq. (9) allows to compare the proposed SBNN with the standard BNN formulation. By setting $\xi = -1/2$ and $\eta = 2$, for which $\alpha = -1$ and $\beta = +1$ (Eq. (4)), and removing the constraint on U leads to the standard formulation of a BNN. This implies that any BNN can be represented using the $\{0,1\}$ domain and perform sparse operations. However, in practice when U is not contrained to be $\leq M$, then $U \approx N/2$ and $h(1/2) = 1$ bit/weight, which means that standard BNNs cannot be compressed more.

3.3 Weight Optimization

In this section, we derive the value of $\Omega = \{\alpha, \beta\}$ which minimizes the quantization error when real-valued weights are quantized using it.

The minimization of the quantization error accounts to minimizing the binarization loss, $\mathcal{L}_B$, which is the optimal estimator when $\widetilde{\mathbf{W}}$ is mapped to $\mathbf{W}$ [30]. This minimization is equivalent to finding the values of α and β which minimize $\mathcal{L}_B$. To simplify the derivation of the optimum α and β values, we minimize $\mathcal{L}_B$

over two variables in one-to-one correspondence with α and β. To achieve this, as in Eq. 4-5, we map $w_i \in \Omega$ to $\overline{w}_i \in \{-1, +1\}$, i.e.

$$w_i = \tau \overline{w}_i + \phi,$$

where τ and ϕ are two real-valued variables, and $\alpha = -\tau + \phi$ and $\beta = \tau + \phi$. As a result, α and β are in one-to-one correspondence with τ and ϕ, and the minimization of $\mathcal{L}_B$ can be formulated as

$$\tau^*, \phi^* = \arg\min_{\tau,\phi} \mathcal{L}_B = \arg\min_{\tau,\phi} \|\widetilde{\mathbf{w}} - (\tau\overline{\mathbf{w}} + \phi\mathbf{1})\|_2 \tag{12}$$

where $\|\cdot\|_2$ is the ℓ_2-norm and $\mathbf{1}$ is the all-one entries matrix.

By first expanding the ℓ_2-norm term and using the fact that $\text{sum}(\overline{\mathbf{w}}) = N^\ell(2p-1)$, it is straightforward to reformulate Eq. 12 as a a function of the sum of real-valued weights, their ℓ_1-norm, the fraction of $+1$-valued binarized weights and the two optimization parameters. In such case, the $\nabla\mathcal{L}_B$ is

$$\nabla\mathcal{L}_B = \begin{pmatrix} \frac{\partial \mathcal{L}_B}{\partial \tau} \\ \frac{\partial \mathcal{L}_B}{\partial \phi} \end{pmatrix} = 2 \begin{pmatrix} -\|\widetilde{\mathbf{w}}\|_1 + N^\ell\big(\tau + \phi(2p-1)\big) \\ -\text{sum}(\widetilde{\mathbf{w}}) + N^\ell\big(\phi + \tau(2p-1)\big) \end{pmatrix}. \tag{13}$$

Solving to find the optimal values τ and ϕ we obtain

$$\tau^* = \frac{\|\widetilde{\mathbf{w}}\|_1}{N^\ell} - \phi^*(2p-1)\,, \quad \phi^* = \frac{\text{sum}(\widetilde{\mathbf{w}})}{N^\ell} - \tau^*(2p-1). \tag{14}$$

When $p = 0.5$, like in standard BNNs, it gives the classical value of $\tau^* = \|\widetilde{\mathbf{w}}\|_1/N^\ell$ as in [30].By substituting ϕ^* in Eq. (12), we obtain the closed-form solution

$$\tau^* = \frac{\|\widetilde{\mathbf{w}}\|_1 - (2p-1)\text{sum}(\widetilde{\mathbf{w}})}{N^\ell(1-(2p-1)^2)}\,, \quad \phi^* = \frac{\text{sum}(\widetilde{\mathbf{w}}) - (2p-1)\|\widetilde{\mathbf{w}}\|_1}{N^\ell(1-(2p-1)^2)}. \tag{15}$$

As the gradient (Eq. 13) is linear in ϕ and τ, this implies that there is a unique critical point. Moreover, an analysis of the Hessian matrix confirms that $\mathcal{L}_B$ is convex and that local minimum is a global minimum. The derivation is here omitted as it is straightforward.

3.4 Network Training

The SBNN training algorithm builds upon state-of-the-art BNN training algorithms [3,15,25], while introducing network sparsification. To profit from BNNs training scheme, we replace $\mathbf{W}_{\{0,1\}}, \xi$ and η (Eq. (10)) with $\overline{W}, \tau$ and ϕ. Doing so, $\mathcal{L}(y, \hat{y})$ corresponds to the loss of BNN algorithms $\mathcal{L}_{\text{BNN}}$. SBNN training also requires to add the penalization term from Eq. (11) to account for sparsity. To account for $\overline{\mathbf{W}}$, the regularization function $g(\mathbf{W}_{\{0,1\}})$ (Eq. (11)) is redefined according to

$$j(\overline{\mathbf{W}}) = \text{ReLU}\left(\left(\sum_i \frac{\overline{w}_i + 1}{2N}\right) - \text{EC}\right), \tag{16}$$

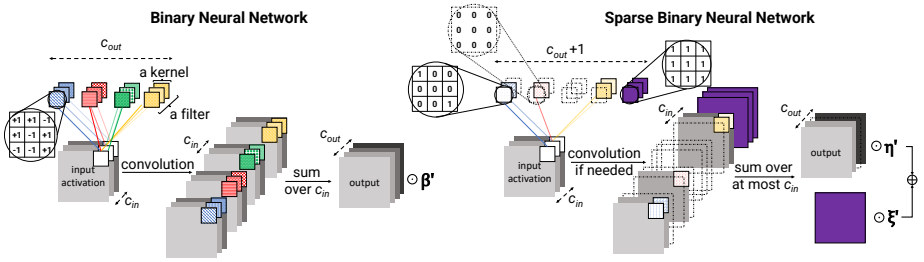

Fig. 1: BNNs vs. SBNNs operations in a convolutional layer using c_{out} filters and input of c_{in} dimensions. BNNs' $(c_{out} \cdot c_{in})$ convolutional kernels are dense and require all computations. SBNNs' kernels are sparse, allowing to skip certain convolutions and sum operations. The removed filters are indicated by a dashed contour and no fill. Both BNNs and SBNNs perform convolutions using XNOR and popcount operations, while the sum is replaced by popcount operations.

and the SBNN objective loss can be expressed as

$$\mathcal{L}_{\text{SBNN}} = \mathcal{L}_{\text{BNN}} + \lambda\, j(\overline{\mathbf{W}}). \tag{17}$$

During training, we modulate the contribution of the regularization term $j(\overline{\mathbf{W}})$ by imposing, at every training iteration, to be equal to a fraction of $\mathcal{L}_{\text{SBNN}}$, i.e.

$$\gamma = \frac{\lambda\, j(\overline{\mathbf{W}})}{\mathcal{L}_{\text{SBNN}}}. \tag{18}$$

The hyperparameter γ is set to a fixed value over all the training process. Since $\mathcal{L}_{\text{SBNN}}$ changes at every iteration, this forces λ to adapt, thus modulating the influence of $j(\overline{\mathbf{W}})$ proportionally to the changes in the loss. The lower γ is set, the less influence $j(\overline{\mathbf{W}})$ has on the total loss. This means that network sparsification will be slower, but convergence will be achieved faster. On the opposite case (high γ), the training will favor sparsification.

3.5 Implementation Gains

We discuss the speed-up gains of the proposed SBNN through its efficient implementation using linear layers in the backbone architecture. Its extension to convolutional layers (Fig. 1) is straightforward, thus we omit it for the sake of brevity.

We describe the use of sparse operations, as it can be done on an FPGA device [7,36]. Instead, when implemented on CPUs, SBNNs can take advantage of pruned layers, kernels and filters for acceleration [9,28,37,38]. Moreover, for kernels with only a single binary weight equal to 1 there is no need to perform a convolution, since the kernels remove some elements from the corner of their input.

The connections in a SBNN are the mapped one-valued weights, i.e. the set S_1. Therefore, SBNNs do not require any XNOR operation on FPGA, being popcount the only bitwise operation needed during the forward pass. The latter, however, is performed only in a layer's input bits connected through the one-valued weights rather than the full input.

For any given layer ℓ, the number of binary operations of a BNN is $\mathcal{O}_{\text{BNN}} = 2N^\ell$ [3], N^ℓ XNOR operations and N^ℓ popcounts. A rough estimate of the implementation gain in terms of the number of binary operations of SBNNs w.r.t. BNNs can be expressed in terms of the EC as

$$\frac{\mathcal{O}_{\text{SBNN}}}{\mathcal{O}_{\text{BNN}}} \approx \frac{2N^\ell}{\text{EC} \cdot N^\ell} \approx \frac{2}{\text{EC}}, \tag{19}$$

which indicates that the lower the EC fraction, the higher the gain w.r.t. BNNs.

Binary operations are not the only ones involved in the inference of SBNN layers. After the sparse $\{0, 1\}$ computations, the mapping operations to the $\{\alpha, \beta\}$ domain take place, also benefiting from implementation gains. To analyze these, let us now denote $\mathbf{x}$ the input vector to any layer and $\mathbf{z} = \mathbf{w}\,\mathbf{x}$ its output. Using E. (4), $\mathbf{z}$ can be computed as

$$\mathbf{z} = \xi\,\mathbf{z}' + \xi\,\eta\,\mathbf{q}, \tag{20}$$

where $\mathbf{z}' = \mathbf{w}_{\{\mathbf{0},\mathbf{1}\}}\,\mathbf{x}$ is the result of sparse operations (Fig. 1), $\mathbf{q} = \mathbf{1}\,\mathbf{x}$, and $\mathbf{1}$ the all-ones matrix.

All the elements in $\mathbf{q}$ take the value $2 \cdot \text{popcount}(\mathbf{x}) - |\mathbf{x}|$, with $|\mathbf{x}|$ the size of $\mathbf{x}$. Therefore, they are computed only once, for each row of $\mathbf{1}$. Being ξ and η known at inference time, they can be used to precompute the threshold in the threshold comparison stage of the implementation of the batchnorm and sign operations following the estimation of $\mathbf{z}$ [35].Thus, SBNNs require $|\mathbf{x}|$ binary operations, one real product and $|\mathbf{x}|$ real sums to obtain $\mathbf{z}$ from $\mathbf{z}'$.

4 Experiments and Results

We first run a set of ablation studies to analyze the properties of the proposed method (Section 4.1). Namely, we analyze the generalization of SBNNs in a standard binary domain and the proposed generic binary domain; we study the role of the quantization error in the network's performance; and the effects of sparsifying binary kernels. Next, we compare our proposed method to other state-of-the-art techniques using the well established CIFAR-10 and CIFAR-100 [18] datasets. Preliminary results on ImageNet [31] are also discussed. All our code has been made publicly available[3].

4.1 Ablation Studies

Experimental setup. We use a ResNet-18 binarized model trained on CIFAR-10 as backbone architecture. We train the networks for 300 epochs, with batch

[3] github.com/robustml-eurecom/SBNN

Table 1: Role of the binary domain and the quantization error when sparsifying BNNs. Experiments performed on CIFAR-10 with a binarized ResNet-18 model.

Domain	Sparsity constraint	Top-1 Accuracy	Δ
Baseline	/	88.93%	/
$\{-\beta, +\beta\}$ [30]	95%	85.95%	-2.98%
$\{\alpha, \beta\}$	95%	86.46%	-2.47%
Learned $\{\alpha, \beta\}$	95%	88.84%	-0.09%

size of 512, learning rate of $1e-3$, and standard data augmentation techniques (random crops, rotations, horizontal flips and normalization). We use an Adam optimizer and the cosine annealer for updating the learning rate as suggested in [24] and we follow the binarization strategy of IR-Net [29].

Generalization properties. We compare the performance of the proposed generic binary domain to other binary domains used by BNNs by assessing the networks' generalization capabilities when the sparsity ratio is 95%. For this experiment, we use the $\{-\beta, +\beta\}$ domain from [30] with no sparsity constraints as the baseline. Additionally, we consider the same domain with a 95% sparsity constraint and the $\{\alpha, \beta\}$ domain obtained optimizing τ and ϕ according to Eq. (15) with the 95% sparsity constraint. Table 1 reports the obtained results in terms of top-1 accuracy and accuracy loss w.r.t. the BNN baseline model (Δ). When we impose the 95% sparsity constraint with the $\{-\beta, +\beta\}$ domain, the accuracy drop w.r.t. to the baseline is 2.98%. Using the $\{\alpha, \beta\}$ domain, the loss goes down to 2.47%, nearly 0.5% better than the $\{-\beta, +\beta\}$ domain. The results suggest that a more general domain leads to improved generalization capabilities.

Impact of the quantization error We investigate the impact of the quantization error in the SBNN generalization. To this end, we compare the proposed quantization technique (Sec. 3.3) with the strategy of learning Ω via backpropagation. We denote this approach Learned $\{\alpha, \beta\}$ (Table 1). The obtained results show that with the learning of the parameters the accuracy loss w.r.t. the BNN baseline decreases down to -0.09%, thus 2.38% better than when τ and ϕ are analytically obtained with Eq. (15). This result implies that the quantization error is one of the sources of accuracy degradation when mapping real-valued weights to any binary domain, but it is not the only source. Indeed, activations are also quantized. Moreover, errors are propagated throughout the network. Learning Ω can partially compensate for these other error sources.

Effects of network sparsification We investigate the effects of network sparsification and how they can be leveraged to reduce the binary operations (BOPs) required in SBNNs. In Section 4.1, we showed that our binary domain is more adept at learning sparse network representations compared to the standard binary domain. This allows us to increase the sparsity of SBNNs while maintaining a desired level of accuracy. When the sparsity is sufficiently high, many convolu-

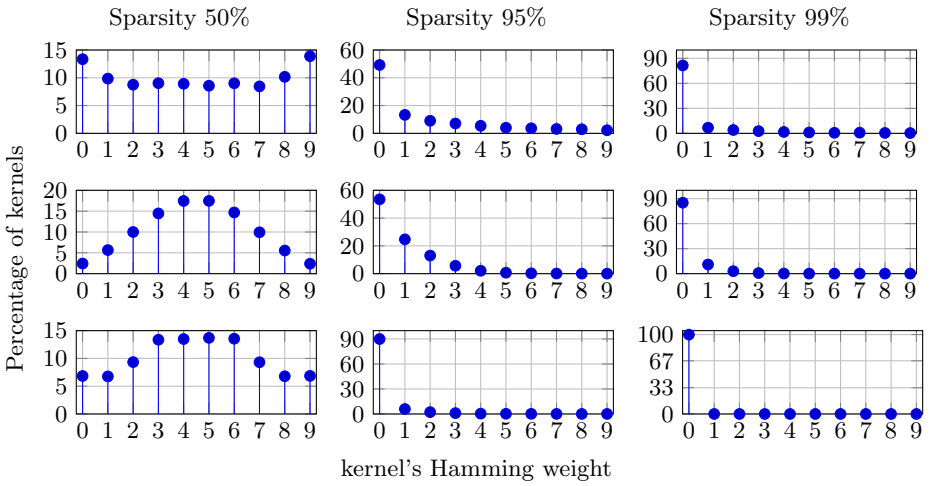

Fig. 2: Percentage of binary kernels for various Hamming weights of a binarized Resnet-18 model over CIFAR-10 for different sparsity constraints. The 5-th, 10-th and 15-th layers are shown in the top, middle and bottom rows, respectively.

tional kernels can be entirely removed from the network, which further reduces the BOPs required for SBNNs. Additionally, convolutional kernels with only a single binary weight equal to 1 do not require a convolution to be performed, as these kernels simply remove certain elements from the input.

To illustrate this effect, we plotted the distribution of binary kernels for the 5th, 10th, and 15th layers of a binarized ResNet-18 model (Fig. 2). The first column shows the distribution when no sparsity constraints are imposed, while the second and third columns show the distribution for sparsity levels of 95% and 99%, respectively. The kernels are grouped based on their Hamming weights, which is the number of non-zero elements in each $\{0,1\}^{3\times3}$ kernel. The plots suggest that increasing the sparsity of SBNNs results in a higher number of kernels with Hamming weights of 0 and 1.

4.2 Benchmark

CIFAR-10. We compare our method against state-of-the-art methods over a binarized ResNet-18 model using CIFAR-10. Namely, we consider: STQ [28], Slimming [37], Dual-P [7], Subbit [36], IR-Net [29] and our method with learned τ and ϕ, for different sparsity constraints. We use the IR-Net as BNN baseline to be compressed. We use the experimental setup described in Sec. 4.1 with some modifications. We extend the epochs to 500 as in [36], and we use a MixUp strategy [42]. In the original IR-Net formulation [29], the training setup is missing. We use our setup to train it, achieving the same accuracy as in [29].

Table 2 reports the obtained results in terms of accuracy (Acc.), accuracy loss w.r.t. the IR-Net model (Δ), and BOPs reduction (BOPs PR). For our SBNN, we estimate BOPs PR by counting the number of operations which are not computed from the convolutional kernels with Hamming weight 0 and 1. For

Table 2: Evaluation of kernel removal for different pruning targets using a binarized Resnet-18 model on CIFAR-10.

Method	Acc.	Δ	BOPs PR	K_0	K_1
IR-Net	91.50%	/	/	/	/
STQ	86.56%	-5.50%	-40.0%	/	/
Slimming	89.30%	-2.20%	-50.0%	/	/
Dual-P (2→1)	91.02%	-0.48%	-70.0%	/	/
Dual-P (3→1)	89.81%	-1.69%	-80.6%	/	/
Dual-P (4→1)	89.43%	-2.07%	-85.4%	/	/
Subbit 0.67-bits	91.00%	-0.50%	-47.2%	/	/
Subbit 0.56-bits	90.60%	-0.90%	-70.0%	/	/
Subbit 0.44-bits	90.10%	-1.40%	-82.3%	/	/
SBNN 50% [**our**]	91.70%	+0.20%	-11.1%	5.6%	6.8%
SBNN 75% [**our**]	91.71%	+0.21%	-24.5%	30.7%	15.9%
SBNN 90% [**our**]	91.16%	-0.24%	-46.5%	61.8%	15.5%
SBNN 95% [**our**]	90.94%	-0.56%	-63.2%	77.1%	11.8%
SBNN 96% [**our**]	90.59%	-0.91%	-69.7%	81.0%	10.1%
SBNN 97% [**our**]	90.71%	-0.79%	-75.7%	84.8%	8.7%
SBNN 98% [**our**]	89.68%	-1.82%	-82.5%	89.3%	6.5%
SBNN 99% [**our**]	88.87%	-2.63%	-88.7%	94.6%	3.3%

other methods, we refer the reader to the original publications. We assess our method at different levels of sparsity, in the range 50 to 99%. For SBNNs we also report the percentage of SBNN's convolutional kernels with Hamming weight 0 (K_0) and with Hamming weight 1 (K_1).

The results suggest that our method is competitive with other more complex pruning strategies. Moreover, our method reports similar accuracy drops w.r.t. state-of-the-art Subbit and Dual-P for similar BOPs PR. However, we need to point out that Subbit and Dual-P results refer to BOPs PR on FPGA, where SBNN can take advantage of sparse operations (Section 3.5) also for the kernels with larger Hamming weights than 0 and 1, because on FPGA all operations involving 0-valued weights can be skipped. For instance, the use of sparse operations on the SBNN 95% allows to remove $\approx 84.9\%$ BOPs.

CIFAR-100. We compare our method in the more challenging setup of CIFAR-100, with 100 classes and 500 images per class, against two state-of-the-art methods: STQ [28], and Subbit [36]. We use ReActNet-18 [25] as the backbone architecture, using a single training step and no teacher. We train for 300 epochs with the same setup used for CIFAR-10 with Mixup augmentation. As no previous results for this setup have been reported for ReActNet-18 and Subbit, for a fair comparison, we trained them from scratch using our setup. We report the same metrics used for CIFAR-10, plus the the reduction of binary parameters (BParams PR). For our SBNN, we estimate BParams PR as follows. For each kernel we use 2 bits to differentiate among zero Hamming weight kernels, one Hamming weight kernels and all the other kernels. Then, we add 4 bits to the kernels with Hamming weight 1 to represent the index position of their 1-valued bit, whereas we add 9 bits for all the other kernels with Hamming weight larger

Table 3: Evaluation of kernel removal for different pruning targets using a ReActNet-18 model on CIFAR-100.

Method	Acc.	Δ	BOPs PR	BParams PR	K_0	K_1
ReActNet-18*	62.79%	/	/	/	/	/
STQ	57.72%	-5.05%	-36.1%	-36.1%	/	/
Subbit 0.67-bits*	62.60%	-0.19%	-47.2%	-33.3%	/	/
Subbit 0.56-bits*	62.07%	-0.72%	-70.0%	-44.4%	/	/
Subbit 0.44-bits*	61.80%	-0.99%	-82.3%	-55.6%	/	/
SBNN 50% [**our**]	63.03%	+0.24%	-11.1%	/	5.6%	6.8%
SBNN 95% [**our**]	63.33%	+0.54%	-66.2%	-59.9%	72.9%	16.6%
SBNN 96% [**our**]	63.04%	+0.25%	-67.3%	-63.7%	78.9%	12.6%
SBNN 97% [**our**]	62.41%	-0.38%	-73.4%	-66.8%	82.9%	11.1%
SBNN 98% [**our**]	63.58%	+0.79%	-79.2%	-70.3%	88.1%	8.0%
SBNN 99% [**our**]	62.23%	-0.57%	-87.8%	-74.0%	93.6%	4.7%

* our implementation.

than 1, which are their original bits. For the other methods, please refer to their work for their estimate of BParams PR.

Table 3 reports the obtained results for the different methods and our SBNN for various sparsity targets. We can see that our pruning method is more effective in reducing both the BOPs and the parameters than Subbit. It allows to remove 79.2% of kernels, while increasing the original accuracy by 0.79% w.r.t. the ReActNet-18 baseline. Instead, we observe nearly 1% accuracy drop for a Subbit network for a similar BOPs reduction. Moreover, our method allows to remove nearly 15% more binary parameters.

ImageNet. We assess our proposed SBNN trained with target sparsity of 75% and 90% on ImageNet. We compare them with state-of-the-art BNNs, namely: XNOR-Net [30], Bi-RealNet-18 [26] and ReActNet-18, ReActNet-A [25] and Subbit [36]. Moreover, we also report the accuracy of the full-precision ResNet-18 [13] and MobileNetV1 [14] models, as a reference. We use a ReActNet-A [25] as SBNN's backbone with its MobileNetV1 [14] inspired topology and with the distillation procedure used in [25], whereas in Subbit [36] they used ReActNet-18 as backbone. One of the limitations of Subbit [36] is that their method cannot be applied to the pointwise convolutions of MobileNetV1 [14]. Due to GPUs limitations, during our training, we decreased the batch size to 64. For a fair comparison, we retrained the original ReActNet-A model with our settings.

Table 4 reports the results in terms of accuracy (Acc). We also include the number of operations (OPs) to be consistent with other BNNs assessment on ImageNet. For BNNs, OPs are estimated by the sum of floating-point operations (FLOPs) plus BOPs rescaled by a factor 1/64 [30,26,25]. We assume sparse operations on FPGA to estimate BOPs for SBNN.

We observe that BOPs are the main contributors to ReActNet-A's OPs (Table 4), thus decreasing them largely reduces the OPs. This, instead, does not

Table 4: Method comparison on ImageNet.

Model	Acc Top-1	BOPs ($\times 10^8$)	FLOPs ($\times 10^8$)	OPs ($\times 10^8$)
MobileNetV1 [14] (full-precision)	70.60	-	5.7	5.7
ResNet-18 [13] (full-precision)	72.12	-	19	19
XNOR-Net [30]	51.20	17	1.41	1.67
Bi-RealNet-18 [26]	56.40	17	1.39	1.63
ReActNet-18 [25]	65.50	17	1.63	1.89
ReActNet-A [25]*	68.12	48	0.12	0.87
Subbit 0.67-bits ReActNet-18	63.40	9	1.63	1.77
Subbit 0.56-bits ReActNet-18	62.10	5	1.63	1.71
Subbit 0.44-bits ReActNet-18	60.70	3	1.63	1.68
SBNN 75% ReActNet-A **[ours]**	66.18	8	0.12	0.25
SBNN 90% ReActNet-A **[ours]**	64.72	2	0.12	0.16

* our implementation.

hold for ReActNet-18, which may explain why Subbit is not effective in reducing OPs of its baseline. Our method instead is effective even for less severe pruning targets and it requires less than 3.4× OPs w.r.t. state-of-the-art ReActNet-A model, while incurring in an acceptable generalization loss between $1.9 - 3.4\%$.

5 Conclusions

We have presented sparse binary neural network (SBNN), a novel method for sparsifying BNNs that is robust to simple pruning techniques by using a more general binary domain. Our approach involves quantizing weights into a general $\Omega = \{\alpha, \beta\}$ binary domain that is then expressed as 0s and 1s at the implementation stage. We have formulated the SBNN method as a mixed optimization problem, which can be solved using any state-of-the-art BNN training algorithm with the addition of two parameters and a regularization term to control sparsity.

Our experiments demonstrate that SBNN outperforms other state-of-the-art pruning methods for BNNs by reducing the number of operations, while also improving the baseline BNN accuracy for severe sparsity constraints. Future research can investigate the potential of SBNN as a complementary pruning technique in combination with other pruning approaches. In summary, our proposed SBNN method provides a simple yet effective solution to improve the efficiency of BNNs, and we anticipate that it will be a valuable addition to the field of binary neural network pruning.

References

1. Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., Ayyash, M.: Internet of things: A survey on enabling technologies, protocols, and applications. IEEE communications surveys & tutorials **17**(4), 2347–2376 (2015)

2. Bello, M.G.: Enhanced training algorithms, and integrated training/architecture selection for multilayer perceptron networks. IEEE Transactions on Neural Networks **3**(6), 864–875 (1992)
3. Bethge, J., Yang, H., Bornstein, M., Meinel, C.: Back to simplicity: How to train accurate bnns from scratch? arXiv preprint arXiv:1906.08637 (2019)
4. Canziani, A., Paszke, A., Culurciello, E.: An analysis of deep neural network models for practical applications. arXiv preprint arXiv:1605.07678 (2016)
5. Courbariaux, M., Bengio, Y., David, J.P.: Binaryconnect: Training deep neural networks with binary weights during propagations. In: Advances in Neural Information Processing Systems (NeurIPS). pp. 3123–3131 (2015)
6. Evans, D.: The internet of things: How the next evolution of the internet is changing everything. CISCO white paper **1**(2011), 1–11 (2011)
7. Fu, K., Qi, Z., Cai, J., Shi, X.: Towards high performance and accurate bnn inference on fpga with structured fine-grained pruning. In: Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design. pp. 1–9 (2022)
8. Gomez, A.N., Zhang, I., Swersky, K., Gal, Y., Hinton, G.E.: Learning sparse networks using targeted dropout. CoRR **abs/1905.13678** (2019), http://arxiv.org/abs/1905.13678
9. Guerra, L., Drummond, T.: Automatic pruning for quantized neural networks. In: 2021 Digital Image Computing: Techniques and Applications (DICTA). pp. 01–08. IEEE (2021)
10. Guo, N., Bethge, J., Meinel, C., Yang, H.: Join the high accuracy club on imagenet with a binary neural network ticket. arXiv preprint arXiv:2211.12933 (2022)
11. Han, S., Mao, H., Dally, W.J.: Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In: International Conference on Learning Representations (ICLR) (2016)
12. Han, S., Pool, J., Tran, J., Dally, W.: Learning both weights and connections for efficient neural network. In: Advances in Neural Information Processing Systems (NeurIPS). pp. 1135–1143 (2015)
13. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
14. Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H.: Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861 (2017)
15. Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., Bengio, Y.: Binarized neural networks. In: Proceedings of the International Conference on Neural Information Processing Systems. pp. 4114–4122 (2016)
16. Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., Bengio, Y.: Quantized neural networks: Training neural networks with low precision weights and activations. The Journal of Machine Learning Research **18**(1), 6869–6898 (2017)
17. Hwang, K., Sung, W.: Fixed-point feedforward deep neural network design using weights+ 1, 0, and- 1. In: IEEE Workshop on Signal Processing Systems (SiPS). pp. 1–6 (2014)
18. Krizhevsky, A., Nair, V., Hinton, G.: Cifar (canadian institute for advanced research). Tech. rep., http://www.cs.toronto.edu/~kriz/cifar.html
19. Kuhar, S., Tumanov, A., Hoffman, J.: Signed binary weight networks: Improving efficiency of binary weight networks by exploiting sparsity. arXiv preprint arXiv:2211.13838 (2022)
20. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**(7553), 436–444 (2015)

21. Liang, T., Glossner, J., Wang, L., Shi, S., Zhang, X.: Pruning and quantization for deep neural network acceleration: A survey. Neurocomputing **461**, 370–403 (2021)
22. Lin, J., Chen, W.M., Lin, Y., Cohn, J., Gan, C., Han, S.: Mcunet: Tiny deep learning on iot devices. In: Conference on Neural Information Processing Systems (NeurIPS) (2020)
23. Lin, X., Zhao, C., Pan, W.: Towards accurate binary convolutional neural network. Advances in neural information processing systems **30** (2017)
24. Liu, Z., Shen, Z., Li, S., Helwegen, K., Huang, D., Cheng, K.T.: How do adam and training strategies help bnns optimization. In: International Conference on Machine Learning. pp. 6936–6946. PMLR (2021)
25. Liu, Z., Shen, Z., Savvides, M., Cheng, K.T.: Reactnet: Towards precise binary neural network with generalized activation functions. In: European Conference on Computer Vision. pp. 143–159. Springer (2020)
26. Liu, Z., Wu, B., Luo, W., Yang, X., Liu, W., Cheng, K.T.: Bi-real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm. In: Proceedings of the European Conference on Computer Vision. pp. 722–737 (2018)
27. Louizos, C., Welling, M., Kingma, D.P.: Learning sparse neural networks through l_0 regularization. In: International Conference on Learning Representations (ICLR) (2018)
28. Munagala, S.A., Prabhu, A., Namboodiri, A.M.: Stq-nets: Unifying network binarization and structured pruning. In: BMVC (2020)
29. Qin, H., Gong, R., Liu, X., Shen, M., Wei, Z., Yu, F., Song, J.: Forward and backward information retention for accurate binary neural networks. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 2250–2259 (2020)
30. Rastegari, M., Ordonez, V., Redmon, J., Farhadi, A.: Xnor-net: Imagenet classification using binary convolutional neural networks. In: Proceedings of the European Conference on Computer Vision. pp. 525–542. Springer (2016)
31. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C., Fei-Fei, L.: ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision (IJCV) **115**(3), 211–252 (2015). https://doi.org/10.1007/s11263-015-0816-y
32. Schiavone, R., Zuluaga, M.A.: Sparse binary neural networks (2021), https://openreview.net/forum?id=SP5RHi-rdlJ
33. Srinivas, S., Subramanya, A., Venkatesh Babu, R.: Training sparse neural networks. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 138–145 (2017)
34. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research **15**(1), 1929–1958 (2014)
35. Umuroglu, Y., Fraser, N.J., Gambardella, G., Blott, M., Leong, P., Jahre, M., Vissers, K.: Finn: A framework for fast, scalable binarized neural network inference. In: Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. pp. 65–74 (2017)
36. Wang, Y., Yang, Y., Sun, F., Yao, A.: Sub-bit neural networks: Learning to compress and accelerate binary neural networks. In: Proceedings of the IEEE/CVF international conference on computer vision. pp. 5360–5369 (2021)
37. Wu, Q., Lu, X., Xue, S., Wang, C., Wu, X., Fan, J.: Sbnn: Slimming binarized neural network. Neurocomputing **401**, 113–122 (2020)

38. Xu, Y., Dong, X., Li, Y., Su, H.: A main/subsidiary network framework for simplifying binary neural networks. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 7154–7162 (2019)
39. Yang, J., Shen, X., Xing, J., Tian, X., Li, H., Deng, B., Huang, J., Hua, X.s.: Quantization networks. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 7308–7316 (2019)
40. Yao, S., Zhao, Y., Zhang, A., Hu, S., Shao, H., Zhang, C., Su, L., Abdelzaher, T.: Deep learning for the internet of things. Computer **51**(5), 32–41 (2018)
41. Zhang, D., Yang, J., Ye, D., Hua, G.: Lq-nets: Learned quantization for highly accurate and compact deep neural networks. In: Ferrari, V., Hebert, M., Sminchisescu, C., Weiss, Y. (eds.) Proocedings of the European Conference on Computer Vision. vol. 11212, pp. 373–390 (2018)
42. Zhang, H., Cisse, M., Dauphin, Y.N., Lopez-Paz, D.: mixup: Beyond empirical risk minimization. In: International Conference on Learning Representations (ICLR) (2018)

Ethical Statement

The proposed SBNN can in principle extend the range of devices, at the edge of communication networks, in which DNN models can be exploited. Our work touches various ethical considerations:

- **Data Privacy and Security**: By performing inference of DNNs directly on edge devices, data remains localized and does not need to be transmitted to centralized servers. This reduces the risk of sensitive data exposure during data transfer, enhancing privacy protection.
- **Fairness and Bias**: SBNNs, like other DNNs at the edge, can be susceptible to biased outcomes, as they rely on training data that may reflect societal biases. However, by simplifying the weight representation to binary values, SBNNs may reduce the potential for biased decision-making because they may be less influenced by subtle variations that can introduce bias. Nevertheless, it is essential to address and mitigate biases in data to ensure fairness in outcomes and avoid discriminatory practices.
- **Transparency and Explainability**: The SBNN design can be applied to DNN models that are designed to provide transparency and explainability. Moreover, the binary nature of SBNNs can make them more interpretable and easier to understand compared to complex, multi-valued neural networks. This interpretability can help users gain insights into the decision-making process and facilitate transparency.
- **Human-Centric Design**: SBNNs can extend the use of DNNs at the edge, extending the range of users of applications which are focused on human well-being, human dignity and inclusivity.
- **Resource Allocation and Efficiency**: SBNNs allows the use of DNNs in a more efficient way from both the use of energy, memory and other crucial resources, thus allowing to reduce the environmental impact of DNNs.
- **Ethics of Compression**: While SBNNs offer computational efficiency and reduced memory requirements, the compression of complex information into

binary values may raise ethical concerns. Compression may lead to over-simplification or loss of critical details, potentially impacting the fairness, accuracy, or reliability of decision-making systems.

It is important to consider these ethical aspects of SBNNs when evaluating their suitability for specific applications and to ensure responsible and ethical deployment in alignment with societal values and requirements.