LAB

Learnable Activation Binarizer for Binary Neural Networks

Falkena, Sieger ; Jamali-Rad, Hadi; van Gemert, Jan

DOI
10.1109/WACV56688.2023.00636

Publication date
2023

Document Version
Final published version

Published in
Proceedings of the 2023 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)

2023 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV) | 978-1-6654-9346-8/23/$31.00 ©2023 IEEE | DOI: 10.1109/WACV56688.2023.00636

LAB: Learnable Activation Binarizer for Binary Neural Networks

Sieger Falkena[1,2], Hadi Jamali-Rad[1,2], Jan van Gemert[1]
[1] TU Delft, Delft, The Netherlands
[2] Shell Global Solutions International B.V., Amsterdam, The Netherlands
sieger.falkena@shell.com, h.jamalirad@tudelft.nl, j.c.vangemert@tudelft.nl

Abstract

Binary Neural Networks (BNNs) are receiving an upsurge of attention for bringing power-hungry deep learning towards edge devices. The traditional wisdom in this space is to employ `sign(.)` *for binarizing feature maps. We argue and illustrate that* `sign(.)` *is a* uniqueness bottleneck*, limiting information propagation throughout the network. To alleviate this, we propose to dispense* `sign(.)`*, replacing it with a learnable activation binarizer (*`LAB`*), allowing the network to learn a fine-grained binarization kernel per layer - as opposed to global thresholding.* `LAB` *is a novel universal module that can seamlessly be integrated into existing architectures. To confirm this, we plug it into four seminal BNNs and show a considerable accuracy boost at the cost of tolerable increase in delay and complexity. Finally, we build an end-to-end BNN (coined as* `LAB-BNN`*) around* `LAB`*, and demonstrate that it achieves competitive performance on par with the state-of-the-art on ImageNet. Our code can be found in our repository:* `https://github.com/sfalkena/LAB`*.*

1. Introduction

Convolutional Neural Networks (CNNs) dominate the current state-of-the-art computer vision tasks. With evolving research, models gained increasingly higher accuracy, but in parallel they have grown in size and complexity. This imposes a significant burden for deploying deep learning models on resource-constrained edge devices. Recent studies explore model compression techniques to reduce model size and latency, such as pruning [25], quantization [36], knowledge distillation [13], neural architecture search [10] and low rank approximation [38]. The most extreme form of quantization is realized by binarization, resulting in binary weights and activations $\{-1, +1\}$. Networks utilizing this are known as Binary Neural Networks (BNNs) and promise a bright future for energy-efficient deep learning. By quantizing weights and activations aggressively, one can theoretically achieve a memory reduction of $32\times$ and a computational speedup of $58\times$ on typical CPUs [31].

The current consensus in literature is to use `sign(.)` as a mapping from the full-precision to binary values. However, this imposes three widely-known issues: (i) the representational power of `sign(.)` with respect to the floating point counterpart decreases from 2^{32} to only 2 information levels [26]; (ii) the derivative of `sign(.)` is a Dirac Delta returning a zero gradient almost everywhere [7]; (iii) `sign(.)` spatially uses the same threshold everywhere, which we further refer to as the global threshold. In this work, we identify and debate about a fourth problem (iv) which we refer to as *uniqueness bottleneck*. Approaching the problem from both qualitative and quantitative angles, we demonstrate that using `sign(.)` further limits the representational capacity of the network. Interestingly, several studies state that `sign(.)` is a sub-optimal binarization operation and that it is not straightforward to find a new binarization function [6, 34]. This is exactly why we embark on this challenge in this paper.

Multiple remedies have been proposed to cope with the aforementioned issues of `sign(.)`, including the introduction of scaling factors [31], gradient approximation [29] and pre-binarization distribution shaping [18]. Amongst these directions, we believe that the pre-binarization reshaping shows the most potential to alleviate the information bottleneck of BNNs. However, in contrast to the existing studies, we argue that shaping the pre-binarization distribution is a means an the end, and not the end in itself. To address the issues enumerated, we design a *learnable* activation binarization function (`LAB`) to automate the mapping from the full-precision feature maps to the binary counterparts, so that the representational capacity of the network (compared to full-precision) is least impacted. This is shown schematically in Fig. 1 (and elaborated in Section 3), where application of a global `sign(.)` threshold on the diverse spectrum of values in the discrete feature maps results in similar looking outputs (acting like a diversity bottleneck), whereas `LAB` can potentially avoid such loss of information and reveal important features for later layers.

2642-9381/23/$31.00 ©2023 IEEE
DOI 10.1109/WACV56688.2023.00636

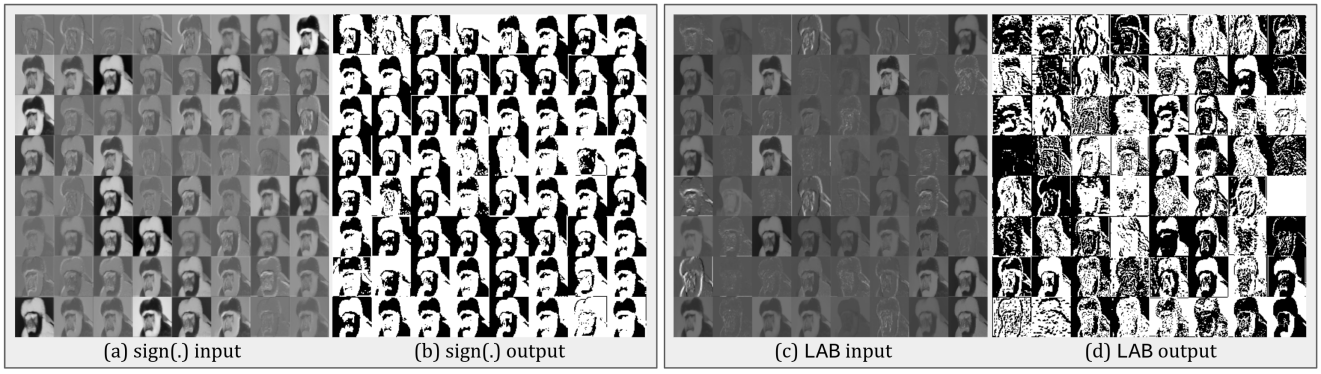

Figure 1: All channels before and after the binarization step for the first binary convolution after model training. In contrast to `sign(.)`, binarization using `LAB` does not map some of the discrete output feature maps into similar binary feature maps, but learns to distinguish important features during binarization, improving the information propagation capacity of the BNN.

Our contributions can be summarized as follows:

- To the best of our knowledge, for the first time, we identify the *uniqueness bottleneck* imposed by the `sign(.)`. We demonstrate that `sign(.)` limits the representational capacity of binary feature maps.
- To address this bottleneck, we introduce a novel learnable activation binarization: `LAB`. We show that `LAB` is a universal module that can readily be plugged into any existing BNN architecture, and improve its performance. Our experimentation on four seminal BNN baselines corroborates this claim.
- We build an end-to-end network around `LAB` (coined as `LAB-BNN`) and demonstrate that it offers competitive accuracy (64.2% Top-1 validation accuracy) on par with the state-of-the-art in this space on ImageNet.

2. Related Work

Current BNNs binarize the full precision weights and activations by applying `sign(.)` on them:

$$x_b = \texttt{sign}\,(x_r) = \begin{cases} +1, \text{ if } x_r > 0 \\ -1, \text{ if } x_r \leq 0 \end{cases}, \qquad (1)$$

where x_b and x_r denote the binary and real (full precision) values, respectively. Naively applying these quantizations to a CNN yields low accuracy and to close the gap between BNN implementations and their real-valued counterparts, several research directions have arisen: minimization of quantization error [7, 31], loss function improvement [9, 16, 24], gradient approximation [21, 24, 29], different network architecture designs [5, 8, 17, 24, 26, 41], training strategies [1, 23, 27, 33] and binary inference engines [3, 12, 37, 39]. Apart from these main directions, a few studies investigate new methodologies for binarizing weights and activations, which will be elucidated next.

Weight binarization. A novel approach for weight binarization is presented in [14] where both full precision and binary weights are employed as noisy supervisors for learning a mapping towards the final binary weights. As this mapping is learnable, it can exploit the relationships between weights. SiMaN [20] and RBNN [21] both propose a new binarization method based on so-called angle alignment between the full-precision and binary weights.

Activation binarization. One way to approach activation binarization is through classic computer vision techniques, such as dithering. This technique can binarize an image in a way that shifts quantization error towards higher frequencies. As the human visual system is more receptive to lower frequencies, the binarized image is perceived as having a low quantization error, and thus, carrying more information. A realization of this idea is called DitherNN but it only reports mild improvement [2]. Most activation binarization approaches focus on shaping the pre-binarization distribution, from which a higher entropy can be achieved after binarization [29]. For instance, an extra regularization term is proposed in [9] to explicitly shape the pre-binarization distribution so that it counteracts degeneration, saturation, and gradient mismatch problems. It is argued in [18] that BNNs benefit from an unbalanced pre-binarization distribution. ReActNet [24] argues that BNNs benefit from learning a similar activation distribution as their full-precision counterparts. On a related note, Si-BNN [34] approaches the activation binarization problem from a somewhat different angle and introduces sparsity in the activation binarization process. Even though these studies show promising performance results for BNNs, we argue that changing the pre-binarization distribution is still a form of adaptive global thresholding, and thus a sub-optimal approach. Therefore, the output feature map does not fully reflect on, and adapt to the local information of the input feature map.

3. Problem Formulation

In this section, we reflect on the limitation a `sign(.)` binarization function inflicts on a BNN. We examine the capability of the network to cope with the single threshold value of `sign(.)`, which we refer to as *global* thresholding - given that its value is the same for all spatial dimensions. In practice, as the input feature map to `sign(.)` is the output of a previous (convolutional) layer, the kernel of that convolution will learn to push or pull parts of the output feature map above or beneath the global threshold value, resulting in the fact that the output feature map will be closely centered around the threshold value. The batch normalization layer can further guide this process by effectively shifting the threshold value. Although this combination is essential for the current learning process of BNNs, we argue that there still is a limitation in its efficacy.

Fig. 2 explains what we perceive as the bottleneck in information propagation of BNNs. Here, $\mathbf{A}$ denotes the discrete input feature map to the binary convolution. Assume $\mathbf{W}$'s represent the set of all unique weight kernels one can imagine. Given a kernel size k, the number of input channels C, and a single output channel per kernel, the total number of unique $\mathbf{W}_i$'s, $\forall i \in [n]$ will then be $n = 2^{k^2 \times C}$. The output feature maps $\mathbf{D}_1$ to $\mathbf{D}_n$ are discrete finite-alphabet tensors. Note that n in this case is smaller than the theoretical maximum number of unique activations $N = (k^2 \times C)^{H \times W}$, given a specific input $\mathbf{A}$. Impacted by the activation design of the previous layers, proper design of $\mathbf{A}$ could potentially minimize the gap between n and N. Applying `sign(.)` on $\mathbf{D}_1$ to $\mathbf{D}_n$ maps them to their binary counterparts $\mathbf{B}_1$ to $\mathbf{B}_n$. In theory, it is possible to have n unique binary feature maps $\mathbf{B}_i$'s, $\forall i \in [n]$, even though in practice, we show the fact that if we give W the freedom to take any value, the set of possible tokens of the output feature map is limited for the `sign(.)`. Low diversity means that multiple distinct values of W will lead to an identical binary output, which will hinder the optimization of the model. We dub the aforementioned issue as the *uniqueness bottleneck*. We argue that due to this bottleneck, the network does not utilize its full potential and the representational capacity of the network is going to be impacted.

To demonstrate that this hypothesis is valid, we design a toy experiment that uses single-layer binary input feature maps $\mathbf{A}$ (in this case with $C = 1$) extracted from the binarized feature maps (using `sign(.)`) in a trained Bi-RealNet-18 [26]. Kernel size k is set to 3, which makes up for a total of $2^{k^2} = 512$ unique kernels. Following the steps sketched in Fig. 2, we take $\mathbf{A}$ as the starting point, convolve it with every possible kernel $\mathbf{W}_i$ and binarize the output activations $\mathbf{D}_i$'s with the `sign(.)` function. We then count the number of unique binary feature maps $\mathbf{B}_i$'s, and average over all the channels (per different layers) of 20 different input images. We denote the ratio of counted unique feature maps (n_c), and theoretical total number of unique feature maps (n_t) as the uniqueness ratio $\eta = n_c/n_t$. The results are shown in Table 1. We can see that going deeper with convolutions, leading to smaller feature maps for layers 9 to 16, the uniqueness ratio decreases and the bottleneck becomes more evident. In the next section, we propose a learnable activation binarizer (`LAB`) as a remedy for this bottleneck.

Table 1: η for 512 unique $\mathbf{W}$'s after applying `sign(.)` function in Bi-Real Net. Later layers show a lower ratio, indicating higher presence of the uniqueness bottleneck.

Layer	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
η	0.964	0.994	0.996	0.998	0.998	0.986	0.991	0.994
Layer	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
η	0.994	0.927	0.943	0.951	0.959	0.747	0.781	0.803

4. The Proposed Method: LAB

One possible approach towards addressing the bottleneck of `sign(.)` binarization is to find a mapping from full-precision activation values to corresponding binary values, in such a way that the embedded spatial information from the input feature map is preserved. To do so, we propose to forge a different path in contrast to the current wisdom of activation distribution shaping [9,18,24]. More concretely, we propose a novel learnable activation binarizer (`LAB`) to *learn* a binarization kernel per layer, as shown in Fig. 3. The figure demonstrates `LAB` as a building block of a standard BNN. Zooming into the `LAB` unit, as we need to apply channel-wise binarization like `sign(.)`, the input is first reshaped for per-channel operations. To capture local spatial information per channel, a 3×3 depthwise convolution with a channel multiplier of 2 is applied. The core idea behind this channel doubling is to construct a *miniature segmentation layer* within the `LAB` unit to classify the input as -1 or $+1$. This classification is done through an `ArgMax(.)` across both channels, reducing the feature map back to a single output channel which is finally reshaped back to its original size. As the `ArgMax(.)` is non-differentiable, we apply the `Soft-ArgMax(.)` for the backward pass [11]. Given that we are dealing with only two classes, the `Soft-ArgMax(.)` in (2) simplifies to a single entry of the `SoftMax(.)` with an extra temperature controlling parameter β which controls the "hardness" of the `ArgMax(.)` approximation:

$$\text{Soft-ArgMax}(x) = \sum_{i=0}^{1} \frac{e^{\beta x_i}}{\sum_j e^{\beta x_j}} i = \frac{e^{\beta x_1}}{\sum_j e^{\beta x_j}}. \qquad (2)$$

The new binarization approach is only used for binarization of the feature maps and not for the binary convolution kernels, because the kernels do not contain enough spatial information for `LAB` to capture.

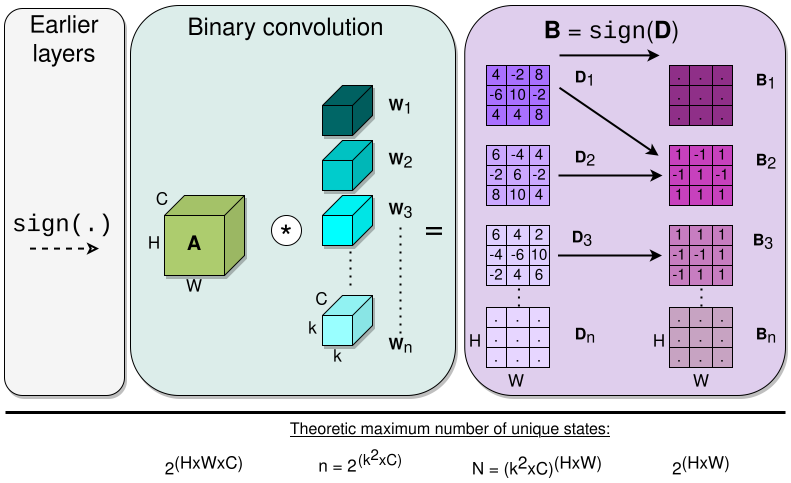

Figure 2: The *uniqueness bottleneck*. Activation $\mathbf{A}$ is convolved with all unique kernels $\mathbf{W}_i$'s. The finite-alphabet feature maps $\mathbf{D}$ are binarized by the `sign(.)`, which creates the bottleneck of multiple $\mathbf{D}$'s mapping to the same binary feature map $\mathbf{B}$. The equations (*at the bottom*) indicate the theoretical maximum number of unique combinations of a tensor.

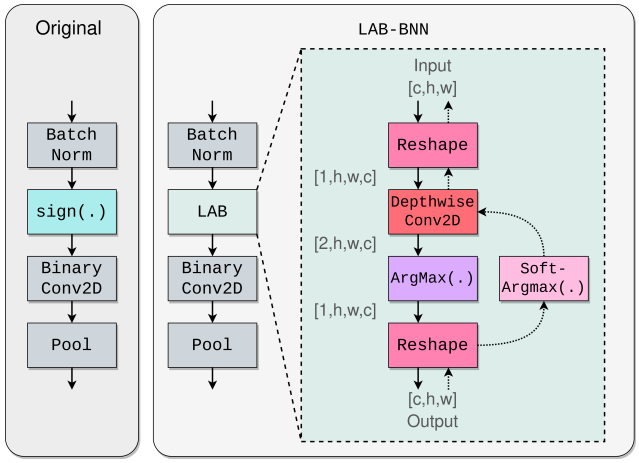

Figure 3: Overview of `LAB` and how it can be used similar to `sign(.)`. Tuples $[\{1,2\}, H, W, C]$ indicate the shape of the tensors. The depthwise convolution together with `ArgMax(.)` form the core of `LAB`. For differentiability in the backward pass , the `Soft-Argmax(.)` is used.

`LAB` ensures that the issues (ii), (iii) and (iv), introduced in section 1 are solved. The learnable depthwise kernel ensures that the binary value of a pixel is dependent its neighbouring pixels. Because of this property, more diverse binary feature maps will be constructed, resulting in lifting of the uniqueness bottleneck. We realize that the introduction of a full-precision depthwise convolution adds additional complexity to the network. However, we argue that the network with `LAB` is still a BNN, as the main convolutions are still binary; the depthwise convolution is an intermediate operation which is more often kept in full-precision in existing literature [20, 21, 25, 27].

5. Experimental Evaluation

In this section, we first reflect on the experimental setup. Next, we analyze the uniqueness bottleneck and how employing `LAB` can help alleviate the problem. We then show that `LAB` is beyond a one-off remedy but more of a universal module that can straightforwardly be plugged into existing baselines. Finally, we propose an end-to-end model architecture (`LAB-BNN`) revolving around `LAB` offering competitive performance against state-of-the-art.

5.1. Experimental Setup

To be able to compare against a variety of existing state-of-the-art baselines - also to allow for a wider adoption by the community - we implement `LAB` both in PyTorch and TensorFlow. For comparison with the state-of-the-art baselines, the TensorFlow-based Larq framework [3] is used.

Network structure. To show the versatility of `LAB`, we plug it into four different architectures. First, XNOR-Net [31], is chosen to examine the improved information flow on an AlexNet-based architecture [19] with no skip connections. Then, Bi-RealNet [26] and ReActNet [24] are used for their ResNet [15] and MobileNet [17] backbones, respectively. Finally, QuickNet [3], an improved version of Bi-RealNet, is used to assess whether `LAB` can make an impact on a top performing state-of-the-art network.

Proposed end-to-end network: `LAB-BNN`. Going beyond the proposed module `LAB`, we also design an end-to-end network based upon the architecture of Bi-RealNet-18[1], which is further enhanced by combining `PRelu` in [7, 24, 27, 33] and initial layers (further referred to as the `STEM`) as proposed by QuickNet [3].

[1]ReActNet-A was tried, but we could not reproduce in Larq.

Hyperparameters. All experiments are conducted using 4 NVIDIA GeForce GTX 1080 Ti GPUs and follow standard settings in Larq [3], unless otherwise mentioned. To reproduce nominal reported performance, we used a batch size of 128 and a learning rate of $1e^{-4}$ for the re-training of XNORNet. For Bi-RealNet a batch size of 256 and learning rate $2.5e^{-3}$ was used. For Quicknet we used a batch size of 512 and a learning rate of $2.5e^{-3}$. Lastly, we re-trained ReActNet-A from scratch with a batch size of 128 and learning rate of $2.5e^{-3}$. `LAB-BNN` uses a batch size of 256 and learning rate of $2.5e^{-3}$ and is trained from scratch for 300 epochs. We chose to train from scratch as recent studies [4] suggest that multi-stage training is not needed for high accuracy models, and that it simplifies the training process significantly. The temperature controlling parameter β of the `Soft-ArgMax(.)` is made a learnable parameter, initialized with the value 1.0.

Real-time inference on the edge. The research in BNNs is focussed on bringing deep learning to *resource-constrained edge* devices. Recent studies report the computational complexity of their models using theoretical metrics such as floating-point operations (FLOPs) [24,27] multiply-accumulate (MACs) [4] or arithmetic computation effort (ACE) [40]. In coherence with [3, 30] we argue that latency is the best metric to compare model performances. In order to benchmark model latency, we use a resource-constrained edge computing device, Nvidia Jetson Xavier NX[2] development kit, which is an ARM-based board for development of embedded AI systems. Although this device has a built-in GPU, for the benchmarking exercise we only use the CPU. Thus, these benchmarks can be reproduced on commodity ARM64 devices. To convert models from TensorFlow to Jetson-compatible models, we use the Larq Compute Engine Converter [3], which outputs a TensorFlow Lite (TFLite [22]) model. This model can now be evaluated using a Larq benchmark tool [3] adapted from the TFLite benchmark[3]. For all the benchmarking experiments, the power mode of the Jetson is set to 15W, we use the single thread mode, and report values averaged over 50 runs.

5.2. The Uniqueness Bottleneck: Qualitative and Quantitative Analysis

In section 3, we have shown that `sign(.)` can introduce a bottleneck in reaching the theoretical maximum number of unique binary states, which we argued would limit the capacity of BNNs. Here, we adopt a small-scale experiment (followed by large-scale end-to-end experiments in the next subsection) to qualitatively and quantitatively demonstrate that `LAB` reduces this bottleneck. To this aim, we compare the binary feature maps of a trained Bi-RealNet-18 with a trained Bi-RealNet-18+`LAB` (where `sign(.)` is replaced with `LAB`). The results are shown in Fig. 4 and Table 2. The figure depicts the selected feature maps from layer 1 and 6 (out of 18 layers). As can be seen, more structural information is preserved in layer 1 of `LAB` compared to `sign(.)`, even though the features become too abstract for human understanding as we go to the deeper layer 6. The most important takeaway from Fig. 4 is to understand that `LAB`, in contrast to `sign(.)` can distinguish between similar pixel values (e.g. shades in the lemons and different tints of blue in the watch). Besides the qualitative demonstration, the impact of `LAB` is further quantified on the number of unique feature maps learned by both networks through two (dis)similarity measures: structural similarity loss (SSIM) [35] and a custom-designed metric we call Euclidean norm dissimilarity (ENDSIM), given by:

Table 2: Dissimilarity comparison between pre- and post-binarization of `sign(.)` vs. `LAB` for SSIM and ENDSIM (3) applied to extracted featuremaps from the trained BiRealNet-18 networks (with and without `LAB`). The direction of the arrow indicates higher dissimilarity. The SSIM values are multiplied by $\times 10^3$.

	Layer	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
SSIM↓ ($\times 10^{-3}$)	**Sign**	94.8	21.3	14.5	13.1	11.9	21.6	16.6	13.5
	LAB	1.5	9.0	4.2	5.4	8.9	5.1	6.6	6.4
ENDSIM↑	**Sign**	1.1	0.93	0.85	0.83	0.8	0.86	0.84	0.8
	LAB	0.92	0.96	1.1	1.3	1.4	1.4	1.5	1.5
	Layer	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
SSIM↓ ($\times 10^{-3}$)	**Sign**	12.2	25.8	19.7	17.6	14.6	29.3	22.4	18.0
	LAB	8.5	4.2	2.9	2.4	8.8	5.6	8.2	7.1
ENDSIM↑	**Sign**	0.76	0.92	0.88	0.85	0.81	0.98	0.93	0.89
	LAB	1.6	1.4	1.4	1.4	1.7	1.4	1.4	1.4

$$ENDSIM(\mathbf{A}_i, \mathbf{A}_j) = \sqrt{\left(\frac{1}{HW}\sum_{w,h}|\mathbf{A}_i - \mathbf{A}_j|\right)^2 + \left(\frac{1}{HW}\sum_{w,h}|\mathbf{A}_i + \mathbf{A}_j|\right)^2}, \quad h \in [H], w \in [W], i \neq j \in [C] \tag{3}$$

where W and H denote the width and height of the input images in pixels and $\mathbf{A}_i$ and $\mathbf{A}_j$, are different single channel feature maps being compared. The metric has two terms in which the first term measures discrepancy, and the second one ensures fully inverted feature maps are not penalized. The latter is because inverted feature map layers along the channels can occur but do not indicate structural differences. Both measures are computed and averaged over all possible combinations of feature map layer pairs across all the 16 convolutional layers of 10 randomly selected images of ImageNet passed through both networks. The results are summarized in Table 2, and as can be seen, except for the first layer, `LAB` shows higher dissimilarity values (a lower SSIM and a higher ENDSIM) indicating more uniqueness along the channels.

[2]https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-xavier-nx/, accessed August 28, 2022

[3]https://www.tensorflow.org/lite/performance/measurement, accessed August 28, 2022

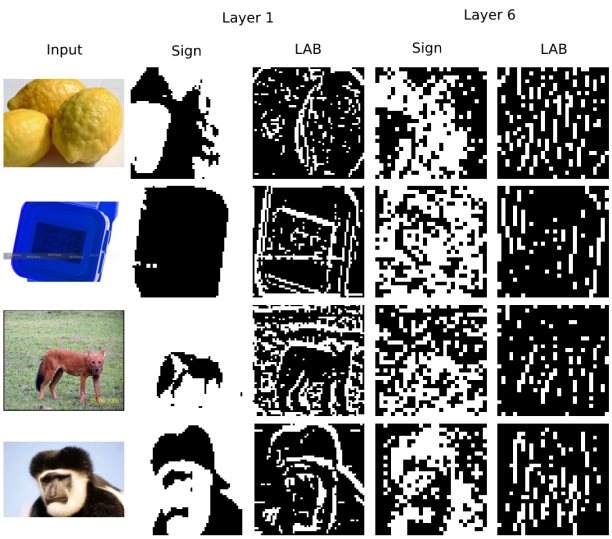

Figure 4: Qualitative comparison between `sign(.)` and `LAB` on two layers of Bi-RealNet-18. `LAB` illustrates higher amount of texture (especially in Layer 1), which indicates allowing more information to pass through.

Table 3: Results of applying `LAB` on the corresponding baselines on ImageNet.

Network	Backbone	Epochs	Method	Top-1 [%]	Top-5 [%]	Model Size [MB]	Latency [ms]
XNOR-Net [31]	AlexNet [19]	60	Sign	44.0	68.1	23.9	50.8
			LAB	46.5	70.3	24.4	55.1
BiRealNet [26]	ResNet-18 [15]	150	Sign	54.4	77.6	4.18	72.5
			LAB	59.1	81.2	4.65	100.6
QuickNet [3]	ResNet-18 [15]	120	Sign	58.7	81.2	4.35	58.1
			LAB	62.5	84.0	4.85	82.4
ReActNet [24]	MobileNetV1 [17]	75	Sign	62.4	83.4	7.74	108.1
			LAB	64.1	84.8	8.69	210.9

5.3. LAB: A Universal Module

The proposed method `LAB` can readily be applied to any BNN, with no architectural adjustments. In this subsection, we demonstrate this by replacing the `sign(.)` with `LAB` in four seminal BNN architectures and evaluate the impact on the downstream classification task of ImageNet. The results are illustrated in Table 3. As can be seen, at the cost of negligible increase in model size and tolerable increase in delay (max of 2x in [ms]), the boosted architectures XNORNet, Bi-RealNet, QuickNet, and ReActNet offer **3.3%**, **7.9%**, **3.8%** and **1.7%** improved Top-1 accuracies, respectively. The performance boost is slightly less pronounced for Top-5 accuracies. Note that the number of required training epochs to reach the nominal performance is dependent upon the architecture itself. The key message from this experiment is that irrespective of the backbone and architecture design, `LAB` makes a considerable impact on all architectures.

Table 4: Comparison against reported state-of-the-art on ImageNet. A dash indicates no value was reported by the original authors.

Network	Method	W/A	Top-1 [%]	Top-5 [%]	BOPs ($\times 10^9$)	FLOPs ($\times 10^8$)	OPs ($\times 10^8$)
ResNet-18	Full-precision	32/32	69.6	89.2	0	18.1	18.1
	Bi-RealNet [26]	1/1	56.4	79.5	1.68	1.39	1.63
	BNN-UAD [18]	1/1	57.2	80.2	-	-	-
	IR-Net [29]	1/1	58.1	80.0	-	-	1.63
	SI-BNN [34]	1/1	59.7	81.8	-	-	-
	SiMaN [20]	1/1	60.1	82.3	-	-	-
	QuickNet [3]	1/1	63.3	84.6	-	-	-
	`LAB-BNN`	1/1	64.2	85.0	1.68	0.66	0.92
	ReActNet [24]	1/1	65.9	-	-	-	-

Table 5: FLOPs comparison: `LAB-BNN` vs. Bi-RealNet.

Component	Operation	FLOPs ($\times 10^6$)
`LAB` (ours)	Depthwise Conv2D	+30.3
	BiasAdd	+1.68
	Multiply	+1.68
	ArgMax	+0.84
	Equal	+0.84
`PRelu` [33]	Mul	+0.57
	Neg	+0.76
STEM [3]	Conv2D	-109.83
Total		**-73.16**

5.4. Comparison Against State-of-the-Art

Now that the universality of `LAB` has been shown, the efficacy of the proposed end-to-end network (`LAB-BNN`) with `LAB` sitting at its core, is evaluated. We compare the performance of `LAB-BNN` against the state-of-the-art baselines focused on activation binarization. For fair comparison, we focus on reported Top-1 and 5 validation accuracies on ImageNet with a ResNet-18 backbone. The outcome is summarized in Table 4. Both Top-1 and 5 results are competitive with the state-of-the-art and come short of the full precision equivalent network by only about 5%. Notably, ReActNet reports a higher accuracy (that we could not reproduce in TensorFlow) in part owing to adopting a multi-stage training strategy, whereas `LAB-BNN` is trained from scratch.

Aside from accuracy, the time and computational complexity of a BNN is just as important, even though most existing baselines overlook the importance of these metrics and sometimes do not even report them. Following [24], in Table 4 we report the total binary and floating-point operations (OP $=$ BOP$/64$ $+$ FLOP). Here, we achieve a lower total computational complexity (in FLOPs) compared to other baselines including the original Bi-RealNet. To further break this down, in Table 5 we set Bi-RealNet as our reference and state the added ($+$) complexity per operation listed in the second column. Even though the mechanics of `LAB` adds slight complexity, we compensate for this with a more efficient implementation of the `STEM` layer [3] leading to an overall decrease of 73.16×10^6 in total FLOPs.

Table 6: LAB−BNN (Imagenette, 100 epochs) with multiple precisions for the depthwise convolution for LAB, and a comparison with other adaptive binarization techniques.

LAB depthwise convolution	**Top 1** [%]	**Top 5** [%]	**Model Size** [MB]
No-LAB	76.4	97.2	2.04
LAB FP32	88.5	99.2	2.30
LAB INT8	89.1	99.2	2.13
LAB INT4	85.3	98.7	2.10

Post-binarization distribution. To illustrate that the post-binarization distribution is not a telling factor about the performance of the network (in contrast to conclusions drawn in [9, 18, 24]), Fig. 5 shows the post-binarization distribution of $+1$'s and -1's for original Bi-RealNet-18 and Bi-RealNet-18+LAB averaged over all channels of 1000 input images. As can be seen, while the distribution of binary values remains almost the same across both networks, the performance of the two (reported in Subsection 5.3) is apart by 7.9%, supporting our claim that the binary distribution is not important, but the spatial arrangement of pixels is.

Going deep or going LAB? Plugging LAB into the standard Bi-RealNet (our base to build LAB−BNN) adds additional complexity to the network, and one could argue that adding the kernel of LAB per layer could virtually help make the network deeper. However, as a counter-argument, we compare LAB−BNN against a Bi-RealNet with a deeper backbone. More specifically, in the current construct, our method adds a depthwise convolution for each binary layer in Bi-RealNet-18. This is (roughly) equivalent to addition of 16 convolution layers, yielding a total of $18 + 16 = 34$ layers. According to [26], Bi-RealNet-34 and Bi-RealNet-50 respectively achieve 62.2% and 62.6% validation accuracies on ImageNet which are both still below the accuracy of LAB−BNN. Furthermore, the model size of BiRealNet-18+LAB is 4.24MB (FP32 weights for depthwise kernel), whereas BiRealNet-34 occupies 5.23MB. Thus, we are offering smaller model size and higher accuracy. To assess the impact of precision on the accuracy gain of LAB, we evaluated the depthwise convolution kernel of LAB for different bit widths. The results summarized in Table 6 indicate that going down from FP32 to INT8, the impact of LAB does not degrade, and even INT4 still offers a significant gain over no-LAB baseline.

Comparison against local thresholding We compare LAB against classical local binarization techniques, to show that LAB adheres to the common wisdom that learnable approaches outperform statistical approaches. We implement Niblack [28] and Sauvola [32] and use a window size of 3 and common values for k. Table 7 shows LAB has a gap of 11.5% with Niblack and improves with 2.7% over the best configuration for Sauvola.

Table 7: LAB−BNN (Imagenette, 150 epochs, batch size 32) A comparison with adaptive binarization techniques.

Binarization method	**Top 1** [%]	**Top 5** [%]	**Model Size** [MB]
STE-sign	87.0	98.7	2.06
Niblack, $k = -0.2$	77.8	97.0	2.06
Sauvola, $k = 0.2$	82.5	98.3	2.06
Sauvola, $k = 0.5$	86.6	98.8	2.06
LAB	89.3	99.1	2.09

6. Ablation Studies

In this section, the effect of different components of LAB−BNN is further inspected. In what follows, we use the same settings listed in Section 5.1, except for the number of epochs which are shortened to 30. We start with the original Bi-RealNet-18 (model **A**) and update the components progressively towards LAB−BNN (model **D**). We first upgrade model **A** by incorporating PRelu activation [33] resulting in **B**. Model **B** is then extended by replacing sign(.) by the proposed LAB module leading to model **C**. Lastly, we replace the initial convolution in Bi-RealNet with the STEM layer of QuickNet [3], which forms LAB−BNN (model **D**). The results are illustrated in Table 8. Plugging LAB into model **B** results in an improvement of about 6% on the Top-1 validation accuracy (and roughly 5% for Top-5), at an increase of 0.46MB in model size and 28.5ms in latency. To make up for the added latency, we apply the STEM layers as proposed by QuickNet [3] which results in decreasing the latency with respect to model **A** down to 9.7ms.

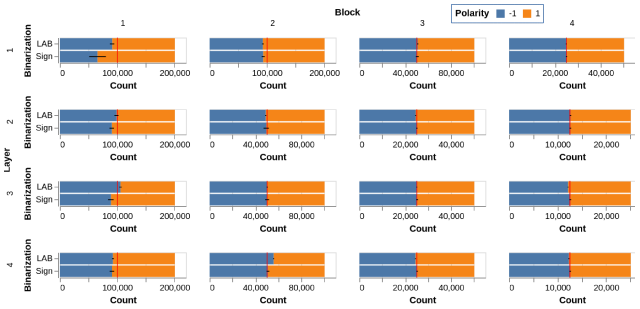

Figure 5: Post-binarization distribution of original Bi-RealNet-18 vs LAB−BNN averaged over all channels of 1000 input images. Rows indicate the blocks in the ResNet structure, columns indicate layer number within each block.

To provide further insights into the distribution of per-operation latencies, in Fig. 6 we profile the network delays for models **A** to **D**. In other words, this is a fine-grained visual breakdown of the total latencies reported in Table 8. The depthwise convolution together with the ArgMax(.) operation in LAB are the main culprits behind the added latency. Additionally, it is clear that the STEM layer indeed helps to alleviate the overall latency as discussed in Table 8.

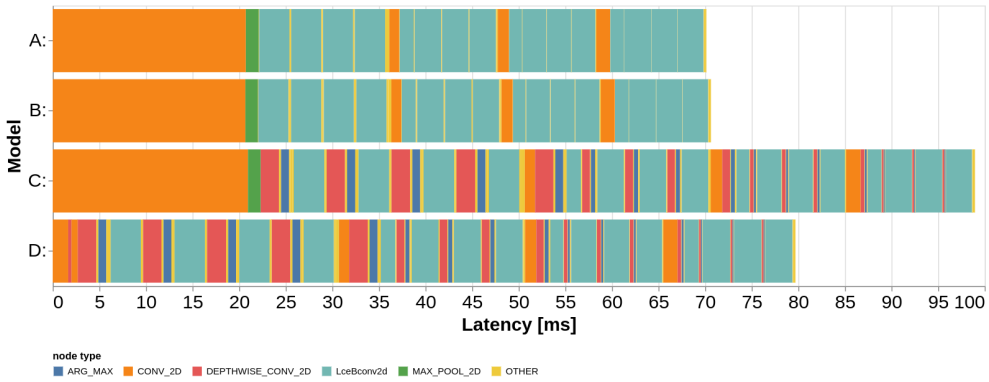

Figure 6: Breakdown of per-operator latencies for ablation study of LAB−BNN. Operators contributing to the latency minimally have been marked as "other".

Table 8: Ablation study for LAB−BNN, trained for 30 epochs on ImageNet dataset. The per-operator breakdown of the latency column is shown in Fig. 6.

Method	Top-1 [%]	Top-5 [%]	Model Size [MB]	Latency [ms]
A: BiRealNet [Base]	45.0	69.7	4.18	70.3
B: Base+PRelu	52.4	76.1	4.19	70.8
C: Base+PRelu+LAB	58.1	80.9	4.65	99.3
D: C+STEM [LAB−BNN]	58.2	80.8	4.62	80.0

6.1. Where to apply LAB.

So far, when applying LAB, all instances of `sign(.)` in every binary layer of the network were replaced with LAB. For the architecture Bi-RealNet-18, this means for all of the four residual blocks, in all four convolutions per block. As we saw in Table 1, the uniqueness bottleneck is mainly visible in later layers. Therefore, in Table 9 we assess in which combination of blocks (out of 16 possible combinations) it is most impactful to apply LAB as a binarization function. In doing so, we apply LAB on all layers per selected block. The networks are trained for 30 epochs and we report Top-1 and Top-5 validation accuracies, model size and and latencies. We conclude from the table that omitting LAB (using the original `sign(.)` everywhere) leads to the worst results, and see that as we apply LAB in more blocks progressively, the accuracy increases, and applying LAB to every block gives the highest accuracy. Interestingly, the model with LAB applied to the last 3 blocks (in the second row) has an exceptional accuracy-latency trade-off. Table 9 shows that by applying LAB in different blocks across the network, it is possible to design a network with a certain accuracy-latency trade-off, which gives useful design freedom for the practical use cases of BNN's.

Table 9: Study on applying LAB to different blocks of LAB−BNN. A checkmark indicates that LAB was used for all layers in that block.

Block 1	2	3	4	Top-1 [%]	Top-5 [%]	Model Size [MB]	Latency [ms]
☑	☑	☑	☑	59.1	81.3	4.68	83.7
☐	☑	☑	☑	58.7	81.0	4.66	68.0
☑	☐	☑	☑	58.4	80.6	4.64	75.3
☑	☑	☑	☐	58.4	80.5	4.33	79.1
☑	☑	☐	☑	58.1	80.4	4.61	77.1
☐	☑	☑	☐	58.0	80.2	4.31	63.9
☐	☑	☐	☑	57.9	80.1	4.58	64.9
☐	☐	☑	☑	57.8	80.0	4.62	60.6
☑	☐	☑	☐	57.8	79.9	4.29	72.9
☐	☐	☑	☐	56.7	78.9	4.27	58.8
☑	☐	☐	☑	56.7	79.3	4.57	71.1
☑	☑	☐	☐	56.4	79.1	4.26	75.0
☐	☐	☐	☑	55.9	78.6	4.54	41.4
☐	☑	☐	☐	55.7	78.1	4.23	47.6
☑	☐	☐	☐	54.9	77.3	4.22	59.9
☐	☐	☐	☐	53.2	76.1	4.20	54.6

7. Concluding Remarks

We have shown that the commonly adopted binarization operation `sign(.)` imposes a *uniqueness bottleneck* on BNNs, making it a sub-optimal choice for binarization. As a remedy, we introduce learnable activation binarizer (LAB), a novel binarization function that allows BNNs to learn a flexible binarization kernel per layer. We have demonstrated that LAB can readily be plugged into existing baseline BNNs boosting their performance regardless of their architecture design. Beyond that, we have also built a new end-to-end network (LAB−BNN) based upon LAB that offers competitive performance on par with the state-of-the-art. For future work, we will investigate applying learnable binarization to weights in addition to activations. We also plan to further extend our experimentation especially around LAB−BNN to push the performance boundaries and advance the state-of-the-art.

References

[1] Milad Alizadeh, Javier Fernández-Marqués, Nicholas D Lane, and Yarin Gal. An empirical study of binary neural networks' optimisation. In *International conference on learning representations*, 2018.

[2] Kota Ando, Kodai Ueyoshi, Yuka Oba, Kazutoshi Hirose, Ryota Uematsu, Takumi Kudo, Masayuki Ikebe, Tetsuya Asai, Shinya Takamaeda-Yamazaki, and Masato Motomura. Dither nn: An accurate neural network with dithering for low bit-precision hardware. In *2018 International Conference on Field-Programmable Technology (FPT)*, pages 6–13. IEEE, 2018.

[3] Tom Bannink, Adam Hillier, Lukas Geiger, Tim de Bruin, Leon Overweel, Jelmer Neeven, and Koen Helwegen. Larq compute engine: Design, benchmark and deploy state-of-the-art binarized neural networks. *Proceedings of Machine Learning and Systems*, 3:680–695, 2021.

[4] Joseph Bethge, Haojin Yang, Marvin Bornstein, and Christoph Meinel. Back to simplicity: How to train accurate bnns from scratch? *arXiv preprint arXiv:1906.08637*, 2019.

[5] Adrian Bulat, Brais Martinez, and Georgios Tzimiropoulos. Bats: Binary architecture search. In *European Conference on Computer Vision*, pages 309–325. Springer, 2020.

[6] Adrian Bulat, Brais Martinez, and Georgios Tzimiropoulos. High-capacity expert binary networks. In *International Conference on Learning Representations*, 2020.

[7] Adrian Bulat, Georgios Tzimiropoulos, Jean Kossaifi, and Maja Pantic. Improved training of binary networks for human pose estimation and image recognition. *arXiv preprint arXiv:1904.05868*, 2019.

[8] Tianlong Chen, Zhenyu Zhang, Xu Ouyang, Zechun Liu, Zhiqiang Shen, and Zhangyang Wang. " bnn-bn=?": Training binary neural networks without batch normalization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4619–4629, 2021.

[9] Ruizhou Ding, Ting-Wu Chin, Zeye Liu, and Diana Marculescu. Regularizing activation distribution for training binarized deep networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11408–11417, 2019.

[10] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *The Journal of Machine Learning Research*, 20(1):1997–2017, 2019.

[11] Chelsea Finn, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine, and Pieter Abbeel. Deep spatial autoencoders for visuomotor learning. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 512–519. IEEE, 2016.

[12] Joshua Fromm, Meghan Cowan, Matthai Philipose, Luis Ceze, and Shwetak Patel. Riptide: Fast end-to-end binarized neural networks. *Proceedings of Machine Learning and Systems*, 2:379–389, 2020.

[13] Jianping Gou, Baosheng Yu, Stephen J Maybank, and Dacheng Tao. Knowledge distillation: A survey. *International Journal of Computer Vision*, 129(6):1789–1819, 2021.

[14] Kai Han, Yunhe Wang, Yixing Xu, Chunjing Xu, Enhua Wu, and Chang Xu. Training binary neural networks through learning with noisy supervision. In *International Conference on Machine Learning*, pages 4017–4026. PMLR, 2020.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[16] Lu Hou, Quanming Yao, and James T Kwok. Loss-aware binarization of deep networks. *arXiv preprint arXiv:1611.01600*, 2016.

[17] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[18] Hyungjun Kim, Jihoon Park, Changhun Lee, and Jae-Joon Kim. Improving accuracy of binary neural networks using unbalanced activation distribution. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 7862–7871, 2021.

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[20] Mingbao Lin, Rongrong Ji, Zihan Xu, Baochang Zhang, Fei Chao, Mingliang Xu, Chia-Wen Lin, and Ling Shao. Siman: Sign-to-magnitude network binarization. *arXiv preprint arXiv:2102.07981*, 2021.

[21] Mingbao Lin, Rongrong Ji, Zihan Xu, Baochang Zhang, Yan Wang, Yongjian Wu, Feiyue Huang, and Chia-Wen Lin. Rotated binary neural network. *Advances in neural information processing systems*, 33:7474–7485, 2020.

[22] TensorFlow Lite. Deploy machine learning models on mobile and iot devices, 2019.

[23] Zechun Liu, Zhiqiang Shen, Shichao Li, Koen Helwegen, Dong Huang, and Kwang-Ting Cheng. How do adam and training strategies help bnns optimization. In *International Conference on Machine Learning*, pages 6936–6946. PMLR, 2021.

[24] Zechun Liu, Zhiqiang Shen, Marios Savvides, and Kwang-Ting Cheng. Reactnet: Towards precise binary neural network with generalized activation functions. In *European Conference on Computer Vision*, pages 143–159. Springer, 2020.

[25] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*, 2018.

[26] 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 *Proceedings of the European conference on computer vision (ECCV)*, pages 722–737, 2018.

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

[28] Wayne Niblack. *An introduction to digital image processing*. Strandberg Publishing Company, 1985.

[29] 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 *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2250–2259, 2020.

[30] Haotong Qin, Xiangguo Zhang, Ruihao Gong, Yifu Ding, Yi Xu, et al. Distribution-sensitive information retention for accurate binary neural network. *arXiv preprint arXiv:2109.12338*, 2021.

[31] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision*, pages 525–542. Springer, 2016.

[32] Jaakko Sauvola and Matti Pietikäinen. Adaptive document image binarization. *Pattern recognition*, 33(2):225–236, 2000.

[33] Wei Tang, Gang Hua, and Liang Wang. How to train a compact binary neural network with high accuracy? In *Thirty-First AAAI conference on artificial intelligence*, 2017.

[34] Peisong Wang, Xiangyu He, Gang Li, Tianli Zhao, and Jian Cheng. Sparsity-inducing binarized neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 12192–12199, 2020.

[35] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[36] Yuhui Xu, Yongzhuang Wang, Aojun Zhou, Weiyao Lin, and Hongkai Xiong. Deep neural network compression with single and multiple level quantization. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[37] Haojin Yang, Martin Fritzsche, Christian Bartz, and Christoph Meinel. Bmxnet: An open-source binary neural network implementation based on mxnet. In *Proceedings of the 25th ACM international conference on Multimedia*, pages 1209–1212, 2017.

[38] Xiyu Yu, Tongliang Liu, Xinchao Wang, and Dacheng Tao. On compressing deep models by low rank and sparse decomposition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7370–7379, 2017.

[39] Jianhao Zhang, Yingwei Pan, Ting Yao, He Zhao, and Tao Mei. dabnn: A super fast inference framework for binary neural networks on arm devices. In *Proceedings of the 27th ACM international conference on multimedia*, pages 2272–2275, 2019.

[40] Yichi Zhang, Zhiru Zhang, and Lukasz Lew. Pokebnn: A binary pursuit of lightweight accuracy. *arXiv preprint arXiv:2112.00133*, 2021.

[41] Baozhou Zhu, Zaid Al-Ars, and H Peter Hofstee. Nasb: Neural architecture search for binary convolutional neural networks. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2020.