BIT-PRUNING: A SPARSE MULTIPLICATION-LESS DOT-PRODUCT

Anonymous authors
Paper under double-blind review

ABSTRACT

Dot-product is a central building block in neural networks. However, multiplication (`mult`) in dot-product consumes intensive energy and space costs that challenge deployment on resource-constrained edge devices. In this study, we realize energy-efficient neural networks by exploiting a `mult`-less, sparse dot-product. We first reformulate a dot-product between an integer weight and activation into an equivalent operation comprised of additions followed by bit-shifts (`add-shift-add`). In this formulation, the number of `add` operations equals the number of bits of the integer weight in binary format. Leveraging this observation, we propose Bit-Pruning, which removes unnecessary bits in each weight value during training to reduce the energy consumption of `add-shift-add`. Bit-Pruning can be seen as soft Weight-Pruning as it prunes bits, not the whole weight element. In extensive experiments, we demonstrate that sparse `mult`-less networks trained with Bit-Pruning show a better accuracy-energy trade-off than sparse `mult` networks trained with Weight-Pruning. (Code will be open-sourced)

1 INTRODUCTION

Modern deep neural networks (DNNs) contain numerous dot-products between input features and weight matrices. However, it is well known that multiplication (`mult`) in dot-product consumes intensive energy and space costs, challenging DNNs' deployment on resource-constrained edge devices. This drives several attempts for efficient DNNs by reducing the energy of `mult`.

Quantization (Yin et al., 2019; Esser et al., 2019; Li & Baillieul, 2004) discretizes the weight and/or activation into a low-bit representation; low-precision `mult` requires less energy than the high-precision counterpart. *Power of two networks* (Li et al., 2019b; Miyashita et al., 2016; Elhoushi) restricts the `mult` operation to the power of two (PoT); PoT `mult` can be realized by a bit-shift (`shift`), which consumes orders of magnitude less energy than `mult`. Although these approaches have achieved remarkable success in reducing energy consumption, they *define* the energy-efficient model structure *before training*. That is, they limit model capacity and impose training with a precision that is difficult to use gradient-based optimization; e.g., approximate gradient with the straight-through estimator (STE) (Yin et al., 2019), making it challenging to achieve good accuracy in the low-bit regime (e.g., 4bit, 2bit, or binary).

Table 1: Energy [pJ] and area [μm^2] cost on ASIC (45nm technology) and FPGA (ZYNQ-7ZC706). Data adapted from (You et al., 2020; Gholami et al., 2021; Horowitz, 2014)

		ASIC		FPGA
Operation	Format	Energy	Area	Energy
`mult`	FIX32	3.1	3495	19.6
	FIX8	0.2	282	0.2
`add`	FIX32	0.1	137	0.1
	FIX8	0.03	36	0.1
`shift`	FIX32	0.13	-	0.1
	FIX8	0.024	-	0.025

Aside from low-bit approaches, *Weight-Pruning* (Frankle & Carbin, 2018; Chen et al., 2022; Yang et al., 2019; Wortsman et al., 2019) *learns* efficient model structure *during training* by removing unimportant weight from high capacity models. Owing to its data-driven nature, it could remove the unnecessary `mult` without sacrificing accuracy (e.g., 95% reduction in unstructured pruning). Empowered by the recent progress of in/near-memory computing architectures (Gholami et al., 2021), several DNN accelerators designed for the unstructured sparsity have emerged (Han et al., 2016; Bamberg et al., 2021; Dietrich et al., 2022; Zhang et al., 2021), realizing the efficient inference

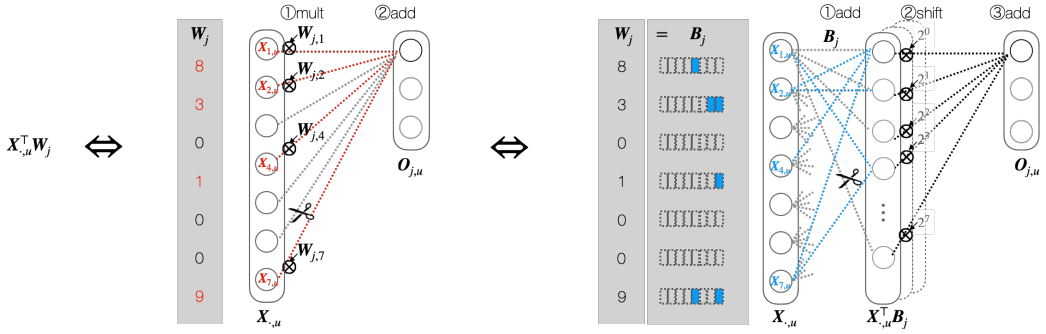


Figure 1: Dot-product realized using sparse `mult-add` (left) and equivalent representation using `add-shift-add`(right). In the `mult-add`, computation is reduced by pruning the entire weight (`mult`). In contrast, in the `add-shift-add` computation is reduced by pruning the bit (`add`) in binary format.

of these sparse networks. Furthermore, optimized implementation for general-purpose CPUs also proved to be a good candidate for unstructured sparsity Kurtz et al. (2020).

Motivated by the significant energy reduction of sparse `mult` operations by Weight-Pruning and recent advances in the frameworks supporting unstructured sparsity, we envision new frontiers in the accuracy/energy trade-off by realizing *sparse* and *`mult`-less* dot-product for hardware supporting the unstructured sparsity. Noting that the dot-product between integer weights and activation can be decomposed into a PoT basis and binary vector, we first reformulate a dot-product between integer weight and activation (`mult-add`) into an equivalent operation comprised of additions followed by bit-shifts and additions (`add-shift-add`, Figure 1). In this formulation, the number of the first `add` operations equals the `bitcount` of the weight elements in binary format. From this observation, we propose Bit-Pruning, which removes unnecessary `add` (therefore unnecessary bits in weight) during training in a data-driven manner to reduce the energy consumption in the `add-shift-add` operation. Because of the difficulty of optimization in binary format, we optimize parameters on a high-precision differentiable `mult-add` network with bit-sparsity regularization, which promotes weights to be sparse in binary format. After training, the obtained bit-sparse `mult-add` network is converted to an equivalent `add-shift-add` network comprised of sparse `add` for efficient inference on DNN accelerators supporting unstructured sparsity.

As depicted in Figure 1, Bit-Pruning can be seen as a soft and fine-grained Weight-Pruning; it does not necessarily remove the whole weight elements but removes only some unnecessary bits of the weight values. This interpretation raises the following question:

> *Which pruning offers a better accuracy/energy trade-off?* `mult` *in* `mult-add` *network (Weight-Pruning) or* `add` *in* `add-shift-add` *network (Bit-Pruning)?*

We conducted an extensive evaluation to answer the question (section 4); the results suggest that pruning the `add` is several times more energy efficient than pruning the `mult`.

Remark: We do not argue that the `add-shift-add` representation itself is efficient. In fact, the estimated energy consumption of (dense) `mult-add` and `add-shift-add` without pruning are almost the same, as we see in section 3.3. Our research interest is to investigate whether the fine-grained pruning by removing bits rather than weights can find a more efficient sparse structure; the `add-shift-add` representation permits this investigation.

2 Preliminaries

2.1 Dot-product with Mult-Add

The dot-product is a fundamental building block of neural networks. In this study, we mainly focus on the dot-product that appears in convolution[1]. Let the weights and activation be quantized to M and N bits, respectively, where M, N are sufficiently fine quantization levels such as $M, N = 8, 32$.

[1]The same discussion can be applied for dot-product in another computation block such as multi-layer perceptron or self/cross attention.

Convolution with the input $\boldsymbol{I} \in \mathbb{Z}^{C_i \times W \times H}$ and weight $\boldsymbol{W} \in \mathbb{Z}^{C_o \times C_i \times k \times k}$ is equivalent to matrix multiplication of unfolded input $\boldsymbol{X} \in \mathbb{Z}^{C_i kk \times WH}$ and the reshaped weight $\boldsymbol{W} \in \mathbb{Z}^{C_o \times C_i kk}$:

$$\boldsymbol{O} = \boldsymbol{I} * \boldsymbol{W} = \boldsymbol{W}\boldsymbol{X}, \tag{1}$$

For brevity, we consider a single input pixel u and an output channel j. Then, the convolution which corresponds to the dot-product of the u-th column vector of the unfolded input $\boldsymbol{X}_{\cdot,u}$ and j-th row vector of the weight $\boldsymbol{W}_j$ is written as follows:

$$\boldsymbol{O}_{j,u} = \boldsymbol{X}_{\cdot,u}^{\top} \boldsymbol{W}_j = \sum_{k=1}^{K} \boldsymbol{X}_{k,u} \cdot \boldsymbol{W}_{j,k}, \tag{2}$$

where $K = C_i kk$. Computing (2) using naive dense `mult-add` requires K `mult` followed by $(K-1)$ `add`, which consumes a lot of energy as it contains many `mult` operations (Table 1).

2.2 Sparse Mult-Add by Weight-Pruning

When a DNN processor supports unstructured weight and dynamic activation sparsity (Han et al., 2016; Bamberg et al., 2021), `mult` can be skipped either when the activation or weight is zero. In this case, the number of `mult` operation become $\sigma_{\texttt{mult}} K$ where $\sigma_{\texttt{mult}}$ is the ratio of the nonzero element defined as $\texttt{bitcount}(\boldsymbol{W}_j \neq 0 \mid \boldsymbol{X}_{\cdot,u} \neq 0)/K$. Therefore, the dot-product using sparse `mult-add` consumes less energy than dense `mult-add`.

The sparse `mult-add` dot-product is typically realized by *Weight-Pruning*, which trains a network with the *weight-sparsity regularization* $\mathcal{L}_{wgt}$ (Liu et al., 2015; Han et al., 2015) [2]:

$$\begin{aligned} \mathcal{L}(\mathcal{W}) &= \mathcal{L}_{task}(\mathcal{W}) + \eta\, \mathcal{L}_{wgt}(\mathcal{W}), \\ \mathcal{L}_{wgt}(\mathcal{W}) &= \sum_{l=1}^{L} \left| \boldsymbol{X}^{(l)} \odot \boldsymbol{W}^{(l)} \right|_0, \end{aligned} \tag{3}$$

where $\mathcal{L}_{task}$ is a loss specific to the given task (e.g., cross-entropy for classification), $\mathcal{W} = \{\boldsymbol{W}^{(l)}\}_{l=1}^{L}$ is a set of weight matrices from all L layers, $\boldsymbol{X}^{(l)}$ is an input of l-th layer, and η is a hyper-parameter to balance the accuracy and computational energy. In practice, l_1-norm or Hoyer(-Square) loss (Yang et al., 2019) is applied to relax l_0-norm for efficient training[3].

3 Bit-Pruning

For more efficient DNN inference, we propose a sparse and `mult`-less dot-product by *softening* the Weight-Pruning. Our intuition is that better energy-accuracy tradeoff could be achieved by bit-level *fine-grained* pruning instead of the weight-level *course* pruning. We first reformulate a dot-product between integer weight and activation into an equivalent operation comprised of `add-shift-add`, which does not contain heavy `mult` (Section 3.1, Figure 1). In this formulation, we can see that the number of the first `add` (which directly translates to energy consumption) equals the `bitcount` of the weight element in binary format. Then, we propose *Bit-Pruning*, which removes unnecessary bits of weight elements during training to achieve efficient inference (Section 3.2).

3.1 Reformulate Dot-product with Add-Shift-Add

An M-bit weight matrix[4] $\boldsymbol{W} \in \mathbb{Z}^{C_o \times K}$ is represented in binary format with a tensor $\boldsymbol{B} \in \{0,1\}^{C_o \times M \times K}$, where $\boldsymbol{B}_{j,\cdot,k} \in \{0,1\}^M$ is the binary representation of the weight value $\boldsymbol{W}_{j,k} \in \mathbb{Z}$:

$$\boldsymbol{W}_{j,k} = \sum_{m=1}^{M} 2^{m-1} \cdot \boldsymbol{B}_{j,m,k}. \tag{4}$$

[2]For $x \in \mathbb{Z}^{I \times J}$ and $y \in \mathbb{Z}^{J \times K}$, $x \odot y \in \mathbb{Z}^{I \times J \times K}$ is defined as $(x \odot y)_{i,j,k} = x_{i,j} \cdot y_{j,k}$.

[3]When the target hardware only supports unstructured weight sparsity (no activation sparsity); the same argument can applied by replacing $\boldsymbol{X}^{(l)}$ with $\mathrm{numel}(\boldsymbol{X})^{(l)}$.

[4]For simplicity, we assume the weight element $\boldsymbol{W}_{j,k}$ is positive in the following explanation.

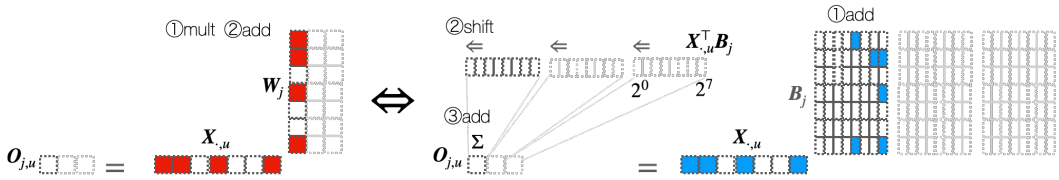


Figure 2: Matrix view of the `mult-add` (left) and `add-shift-add` (right) dot-product.

Then, we rewrite the `mult-add` dot-product (2) into the following `add-shift-add` formulation:

$$\begin{aligned} \boldsymbol{X}_{\cdot,u}^\top \boldsymbol{W}_j &= \sum_{k=1}^{K} \left(\boldsymbol{X}_{k,u} \cdot \sum_{m=1}^{M} 2^{m-1} \cdot \boldsymbol{B}_{j,m,k} \right) \\ &= \sum_{m=1}^{M} 2^{m-1} \cdot \left(\boldsymbol{X}_{\cdot,u}^\top \boldsymbol{B}_{j,m} \right). \end{aligned} \tag{5}$$

This is computed first by accumulating the input $\boldsymbol{X}_{\cdot,u}$ along the binary vector $\boldsymbol{B}_{j,m}$ using `add`, then multiplying 2^m using `shift`, and finally aggregating the results of all M bits by `add`. When the operation is mapped into the hardware supporting the unstructured sparsity, the first `add` using the binary vector $\boldsymbol{B}_{j,m}$ requires `bitcount`$(\boldsymbol{B}_{j,m})$ times `add`. The `shift` and last `add` are executed for $M-1$ times (Table 2).

3.2 Sparse Add-Shift-Add by Bit-Pruning

In the `add-shift-add` formulation of (5), the number of the first `add` operations equals the `bitcount` of the binary tensor $\boldsymbol{B}$, which directly translates to energy consumption. Therefore, our goal is to learn the sparse binary tensor of the `add-shift-add` network that also achieves good performance by minimizing the following loss:

$$\mathcal{L}(\mathcal{B}) = \mathcal{L}_{task}(\mathcal{B}) + \eta \ \texttt{bitcount}(\mathcal{B}), \tag{6}$$

where $\mathcal{B} = \{\boldsymbol{B}^{(l)}\}_{l=1}^{L}$ is a set of binary tensors from all L layers. However, binary parameters prohibit gradient computation, making it difficult to train with standard backpropagation.

Instead of directly optimizing the binary tensor $\boldsymbol{B}$, as done by Yang et al. (2021), we deal with the original weight matrix $\boldsymbol{W}$ while promoting it to be sparse in the binary format $\boldsymbol{B}$. Notice that any dot-product with the integer weight can be precisely expressed as the `add-shift-add` dot product, and the number of `add` counts for $\boldsymbol{B}$ depends on the corresponding integer weight $\boldsymbol{W}$. For example, when $\boldsymbol{W}_{j,k}$ is zero, then $\boldsymbol{B}_{j,\cdot,k} = [0,0,0,0,0,0,0]^\top$, and there is no `add`. When $\boldsymbol{W}_{j,k}$ is PoT (e.g., 2^3), then $\boldsymbol{B}_{j,\cdot,k} = [0,0,1,0,0,0,0]^\top$, and there is only one `add`. In such a way, a number of `add` in `add-shift-add` dot-product depends on the value of the corresponding weight $\boldsymbol{W}_{j,k}$ (Figure 3a), and there is *favorable* value in terms of energy consumption.

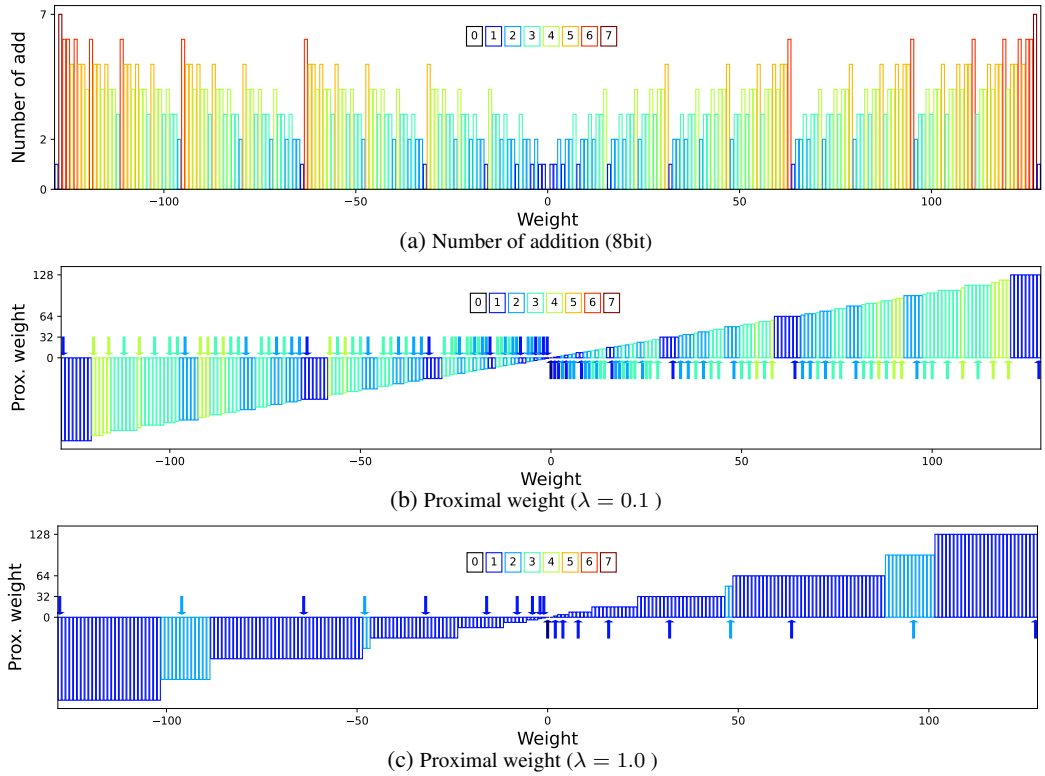
We leverage this to propose a framework called Bit-Pruning, which optimizes the network's energy consumption in a data-driven manner. The binary tensor $\boldsymbol{B}$ of the `add-shift-add` network is trained as the high-precision `mult-add` network having weight $\boldsymbol{W}$ (Figure 2), and a sparsification of $\boldsymbol{B}$ is realized by imposing the *bit-sparsity regularization* $\mathcal{L}_{bit}$ on weight $\boldsymbol{W}$:

$$\begin{aligned} \mathcal{L}(\mathcal{W}) &= \mathcal{L}_{task}(\mathcal{W}) + \eta \ \mathcal{L}_{bit}(\mathcal{W}), \\ \mathcal{L}_{bit}(\mathcal{W}) &= \sum_{l=1}^{L} \left| \boldsymbol{X}^{(l)} \odot (\boldsymbol{W}^{(l)} - \hat{\boldsymbol{W}}^{(l)}) \right|_0, \end{aligned} \tag{7}$$

where $\mathcal{L}_{bit}$ measures a deviation of the current weight $\boldsymbol{W}$ from bit-sparse *proximal weight* $\hat{\boldsymbol{W}}$, explained as follows. Similar to Weight-Pruning (3), we relax the l_0-norm in (7) with l_1-norm or Hoyer(-Square) loss in practice.

Proximal weight Given the current weight $\boldsymbol{W}$, the proximal weight $\hat{\boldsymbol{W}}$ is computed (element-wise) as a minimizer of the sum of *weight proximity cost* $\mathcal{C}_{mv}$ and `add` *count cost* $\mathcal{C}_{add}$:

$$\hat{\boldsymbol{W}}_{j,k} = \arg\min_{w} \left(\mathcal{C}_{mv}(w, \boldsymbol{W}_{j,k}) + \lambda^{(l)} \mathcal{C}_{add}(w) \right), \tag{8}$$



(a) Number of addition (8bit)

(b) Proximal weight ($\lambda = 0.1$)

(c) Proximal weight ($\lambda = 1.0$)

Figure 3: (a) Number of `add` for each weight (`add` count cost $\mathcal{C}_{add}$). (b)-(c) Proximal weight $\hat{\boldsymbol{W}}$ (arrow), which is a solution of (8) (number of `add` for `add-shift-add` dot-product is encoded as color). Refer to Appendix C for the visualization of corresponding loss landscape ($\mathcal{C}_{mv}(w, \hat{w}) + \mathcal{C}_{add}(\hat{w})$).

where $\mathcal{C}_{add}(w)$ is the `bitcount` of w in binary format (Figure 3a) and $\mathcal{C}_{mv}(w, w')$ measures the proximity of two weight values w and w'. In this study, we adopted $\mathcal{C}_{mv}(w, w') := |\mathrm{sgn}(w)|w|^p - \mathrm{sgn}(w')|w'|^p|$. We choose $p = 1/2$, to discount the change in $\mathcal{C}_{mv}$ for a larger value (assuming accuracy is less affected by the change when it is large). Intuitively, $\mathcal{C}_{add}$ encourages $\hat{\boldsymbol{W}}_{j,k}$ to be sparse in the binary format while $\mathcal{C}_{mv}$ keeps it close to the current weight $\boldsymbol{W}_{j,k}$ and $\lambda^{(l)}$ controls its balance (larger λ induce more sparsity). Owing to the discrete structure of the weight space (e.g., 8 bit), the solution of (8) can be analytically computed for given λ (Figure 3b-3c).

Efficient sparse inference. Once the training has been completed, the high-precision `mult-add` network is converted into a mathematically equivalent `add-shift-add` network for efficient inference. It is expected to be comprised of sparse `add` which consumes less energy.

3.3 Which is efficient? Weight-Pruning or Bit-Pruning

An important question in this study is whether pruning the `add` (Bit-Pruning) is more efficient than pruning the `mult` (Weight-Pruning). When considering the dot-product between two K-dimensional, M-bit vectors, the energy consumption of sparse `add-shift-add` ($E_{\texttt{add-shift-add}}$) over that of sparse `mult-add` ($E_{\texttt{mult-add}}$) is calculated from Table 2:

Table 2: Number of operations for dot-product between K-dimensional M-bit vectors.

	`mult-add`		`add-shift-add`
	`dense`	`sparse`	`sparse/dense`
`mult`	K	$\sigma_{\texttt{mult}} K$	0
`add`	$K-1$	$\sigma_{\texttt{mult}} K$	$M\sigma_{\texttt{add}}K+M-1$
`shift`	0	0	$M-1$

$$\frac{E_{\texttt{add-shift-add}}}{E_{\texttt{mult-add}}} = \frac{(M\sigma_{\texttt{add}}\, K)E_{\texttt{add}} + (M-1)E_{\texttt{shift}} + (M-1)E_{\texttt{add}}}{(\sigma_{\texttt{mult}}\, K)E_{\texttt{mult}} + (\sigma_{\texttt{mult}}\, (K-1))E_{\texttt{add}}} \approx \frac{ME_{\texttt{add}}}{E_{\texttt{mult}}}\frac{\sigma_{\texttt{add}}}{\sigma_{\texttt{mult}}} \quad (9)$$

where $\sigma_{\texttt{mult}}$ and $\sigma_{\texttt{add}}$ are the ratio of nonzero elements in weight ($\boldsymbol{W}_j$) and the ratio of nonzero bits in its binary format ($\boldsymbol{B}_j$), respectively. $E_{\texttt{mult}}$, $E_{\texttt{add}}$, and $E_{\texttt{shift}}$ are energy consumption of corresponding operations and we assume $K \gg M$ in the last approximation. In the case of ASIC, we

Table 3: Experimental setup

	CIFAR-10	CIFAR-100	ImageNet
Network	ResNet18	ResNet18	ConvNeXt-B
Batch size	512		256
Training epochs	200		100
Optimizer	AdamW (Loshchilov & Hutter, 2019)		
Scheduler	OneCycle (Smith & Topin, 2019)		Cosine decay (Loshchilov & Hutter, 2017)
Weight quantization M	8		
Activation quantization N	4/8/32		8
Weight initialization	Kaiming-uniform (He et al., 2015)		Pretrained[5]
$\lambda^{(l)}$ for $\mathcal{C}_{move}$ (Bit-Pruning only)	1.0 for all layers (Figure 3c)		

roughly get $E_{\texttt{mult}} \approx ME_{\texttt{add}}$ from Table 1; therefore, the computational efficiency of Bit-Pruning over Weight-Pruning is almost determined by the ratio of obtained connection density $\sigma_{\texttt{add}}/\sigma_{\texttt{mult}}$.

We expect that the $\sigma_{\texttt{add}}$ in a Bit-Pruned network is much smaller than the $\sigma_{\texttt{mult}}$ in a Weight-Pruned network when both networks achieve comparable accuracy. This is because Bit-Pruning removes connections more finely than Weight-Pruning; in other words, given the same nonzero ratio, i.e., $\sigma_{\texttt{add}} = \sigma_{\texttt{mult}}$, Bit-Pruned weight *can* have a more nonzero element (at most M times) than Weight-Pruned weight when it learns a weight that is sparse in binary format. In the extreme case where all the nonzero weights in a network are represented as PoT, the computational cost in `add-shift-add` representation is M times smaller than that in `mult-add` representation. The evaluation on actual DNN training is presented in Section 4.

3.4 Bit-Pruning as a unified low-energy model

(Non-uniform) quantization. Bit-Pruning effectively applies different non-uniform mixed-precision quantization for *each weight element*. As a result of the Bit-Pruning, we will get a sparse $\boldsymbol{B}$; this has a similar effect to the non-uniform quantization of weight. Existing quantization techniques learn (non-uniform) step size for a *group of weights*; contrary, Bit-Pruning realizes pseudo-non-uniform quantization by *selecting a bit pattern*.

PoT dot-product. Bit-Pruning can be interpreted as the soft version of the (additive) PoT quantization. The bit-sparsity regularization prefers sparse binary representation, which corresponds to the PoT; however, it could learn to choose individual non-PoT values for each weight element when necessary (i.e., when it achieves less total energy for the same accuracy). The learned weight by Bit-Pruning is *dominated by PoT, while some have multiple nonzero bits* (See Figure 6).

Unstructured Weight-Pruning. Bit-Pruning can be seen as a soft, fine-grained version of Weight-Pruning. Both bit-sparsity and weight-sparsity regularization loss are minimized when $\boldsymbol{W} = \boldsymbol{0}$; this corresponds to zero `add` in `add-shift-add` and zero `mult` in `mult-add`. However, bit-sparsity regularization does not necessarily remove the *whole weight elements* but only *some unnecessary bits* in the weight element.

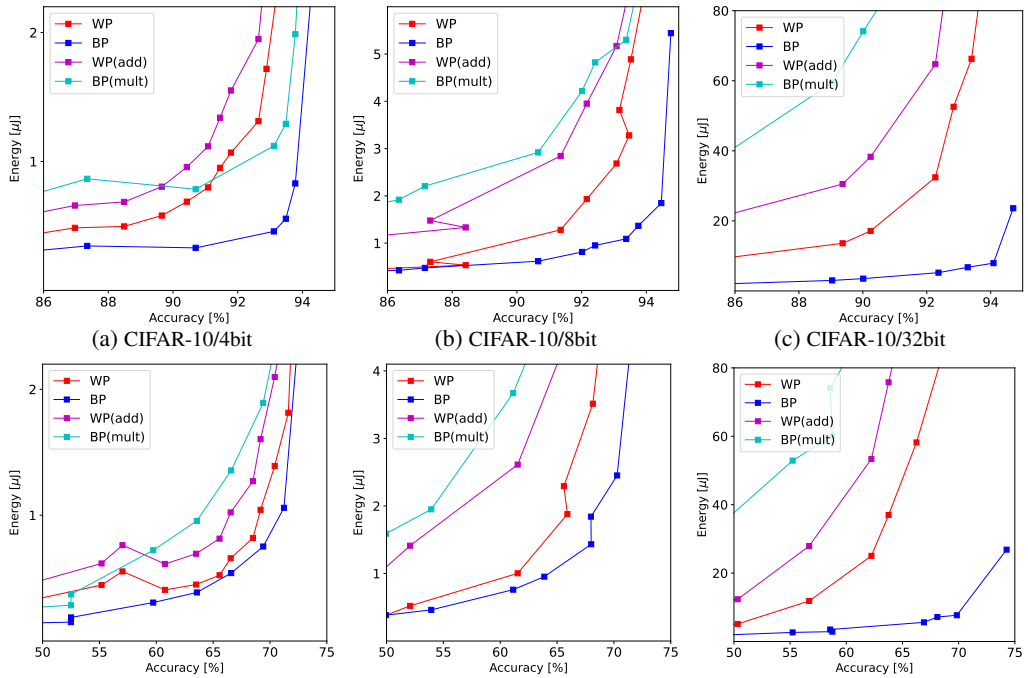
4 Experiment

4.1 Evaluation of accuracy/energy trade-off

Which is energy efficient; pruning `add` *(Bit-Pruning) or pruning* `mult` *(Weight-Pruning)?* We evaluated the accuracy/energy trade-off of both methods on CIFAR-10 (Krizhevsky et al., a), CIFAR-100 (Krizhevsky et al., b), and ImageNet (Deng et al., 2009). For both bit-sparsity regularization $\mathcal{L}_{bit}$ (7) and weight-sparsity regularization $\mathcal{L}_{wgt}$ (3) which is used for all the conv layer, we approximate l_0-norm using Hoyer-Square loss (Yang et al., 2019). We use the same network architecture, quantization method (we use LSQ Esser et al. (2019)), and training strategy except for the loss function for both frameworks. The primary experimental setup[6] is summarized in Table 3. To compare the trade-off, we train networks across various sparsity coefficients η of $\mathcal{L}_{wgt}$ and $\mathcal{L}_{bit}$ (fixed across the entire training sequence).

[5]Pytorch official model zoo `pytorch.org/vision/main/models/resnext`

[6]See appendix for more detailed experimental setup (Appendix A) and additional results (Appendix B).



(a) CIFAR-10/4bit (b) CIFAR-10/8bit (c) CIFAR-10/32bit

(d) CIFAR-100/4bit (e) CIFAR-100/8bit (f) CIFAR-100/32bit

Figure 4: Accuracy/energy trade-off of ResNet18 trained on CIFAR-10 and CIFAR-100 for difference activation quantization level. Bit-Pruning (`add-shift-add`) (BP) and Weight-Pruning (`mult-add`) (WP) are compared. Bit-Pruning always uses an 8bit width binary tensor $\boldsymbol{B}$, and Weight-Pruning quantizes weight using the same quantization level for activation. The result of Weight-Pruning in `add-shift-add` representation (WP(add)) and Bit-Pruning in `mult-add` representation (BP(mult)) is provided for reference. The multiply-accumulate (MAC) of the original dense `mult-add` network is about 0.55×10^6, and the estimated energy consumption (from Table 1) are 26μJ, 107μJ, and 1721μJ for 4bit, 8bit, and 32bit network, respectively.

CIFAR-10/CIFAR-100. We use ResNet18 (He et al., 2016a) as one of the most popular architectures and train them from scratch. Figure 4 summarizes the results on CIFAR-10 (Figure 4a,4b,4c) and CIFAR-100 (Figure 4d,4e,4f).

ImageNet. We conducted an experimednt using pre-trained ConvNeXt (Liu et al., 2022) on ImageNet (Deng et al., 2009). The ConvNeXt is designed to be energy efficient by its network architecture; c.f., it heavily utilizes depthwise convolutions to reduce the number of `mult`. Results are summarized in Figure 5.

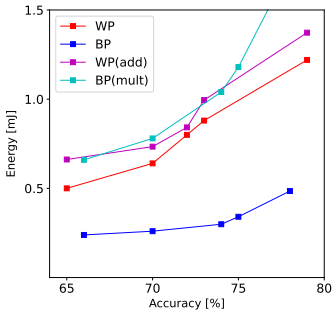
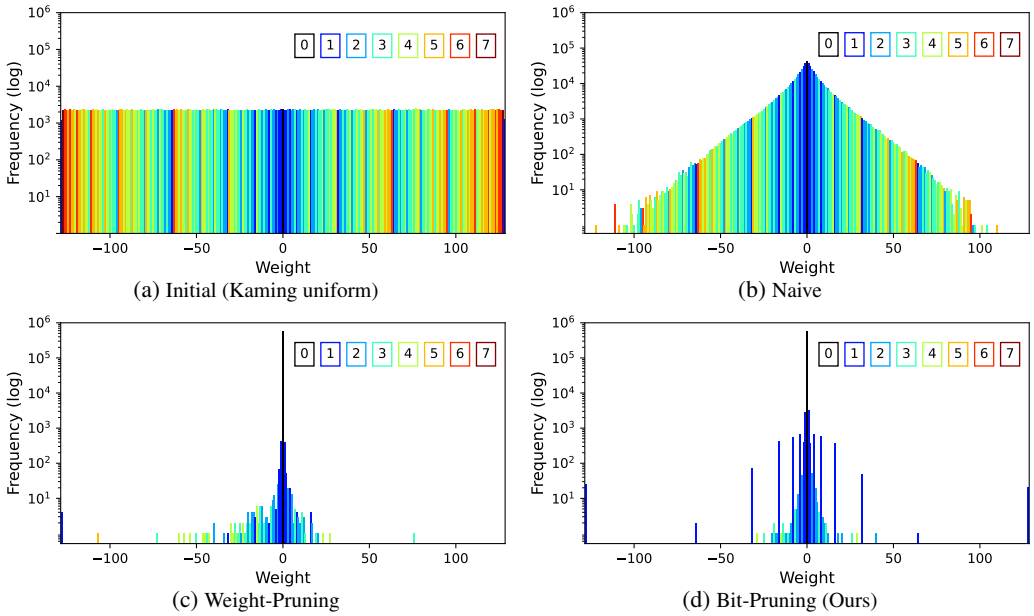


Figure 5: Result of ConvNeXt-B on ImageNet. 8bit activation. See Figure 4 for legend.

Summary. In all ranges of accuracy, `add-shift-add` network trained with Bit-Pruning consumes less energy than that of `mult-add` network trained with Weight-Pruning. In particular, huge energy saving is observed in high accuracy regime, demonstrating the superior data-adaptivity of Bit-Pruning over Weight-Pruning. We see a similar trend for the large-sized, already efficient networks (ConvNeXt) on a large-scale dataset (ImageNet). The number of `mult` in `mult-add` network and `mult` in `mult-add` can be compared by multiplying M (number of bits for activation) to the results in the plots (Because we have $E_{\texttt{mult}} \approx ME_{\texttt{add}}$ from Table 1, Section 3.3). The number of `add` in `add-shift-add` (from Bit-Pruning) is larger than `mult` in `mult-add` (from Weight-Pruning), however; this is significantly less than $M\times$; therefore we'll get the better trade-off.

Notice that the Bit-Pruned network does not show efficiency in `mult-add` representation (BP(mult)), and the Weight-Pruned network does not show efficiency in `add-shift-add` representation (WP(add)). This is because the Bit-Pruned network prefers sparse weights in binary format (Figure 6), which is not necessarily zero, and Weight-pruned network prefer small weight which is not necessarily sparse in binary format.



(a) Initial (Kaming uniform)
(b) Naive
(c) Weight-Pruning
(d) Bit-Pruning (Ours)

Figure 6: Weight histogram (CIFAR-10/ResNet18 13th convolution layer). *Naive* is a model trained using only task loss (test accuracy: 93.1%). We choose a model performing around 93.0% test accuracy for Weight-Pruning and Bit-Pruning. The color indicates the number of addition for `add-shift-add`.

4.2 Ablation on base bit width

Can we obtain a better accuracy-energy trade-off by pruning bits from a wider bit tensor? We conducted the accuracy/energy trade-off comparison between Bit-Pruning with different base bit widths (4bit/8bit) (Figure 7). The result shows the network with a wide tensor (8bit) shows a better trade-off than one using a narrow tensor (4bit). We consider it is because a network with a wide bit has more choices for selecting *good* PoT values, suggesting that a better bit-sparse network is obtained from the larger design space, just like the case of Weight-Pruning (Frankle & Carbin, 2018).

4.3 Learned weight distribution

Figure 6 visualizes histograms of learned weight. Starting from the initial uniform distribution, both Bit-Pruning and Weight-Pruning drive most of the weight to zero; this corresponds to removing `mult` in the `mult-add` network and `add` in the `add-shift-add` network, respectively. In Bit-Pruning, most of the remaining nonzero weights are concentrated on sparse values in a binary format, such as PoT. Whereas, Weight-Pruning did not show such particular preference.

5 Related Works

ShiftAddNet (You et al., 2020) is `mult`-less DNNs utilizing `shift` and `add` for efficient training and inference. It realizes a better accuracy/energy trade-off than the former study utilizing only `add` (AdderNet (Chen et al., 2020)). Ours and ShiftAddNet have similarities in that both utilize `shift` and `add` to realize `mult`-less DNNs. However, their computational model is fundamentally different from ours; ShiftAddNet computes the interaction between weight and input by first computing the dot-product with PoT weight followed by the l_1 norm with `add` weight. It does not have an equivalent `mult-add` dot-product. Its limited expressiveness and the non-differentiability of the PoT operation make training harder (requiring specifically designed gradient computation) and prohibit it from reaching the accuracy of the corresponding `mult-add` network. Ours can be learned as an ordinal high-precision differentiable `mult-add` network; therefore, it is possible to reach the performance of the original network. Besides, motivation and target hardware is different; ShiftAddNet aims to realize efficient training and inference on dense vector-type processors, while

our target is hardware supporting unstructured sparsity solely for inference[7]. Instead, using the recent extension, ShiftAddNAS Huihong Shi (2022), ours and ShiftAddNet can be combined, e.g., the best combination of `shift-add` and `add-shift-add` are automatically searched.

Lottery ticket hypothesis states that *"dense, randomly-initialized, feed-forward networks contain sub-networks that have the equivalent accuracy as the original network (winning tickets): unstructured weight pruning naturally uncovers the winning tickets"* (Frankle & Carbin, 2018; Ramanujan et al., 2020). The comparison of different bit widths in Bit-Pruning (Section 4.2) might indicate the *lottery ticket hypothesis* in *bits*; a network having more connections in terms of `add-shift-add` representation (i.e., the large binary tensor $\boldsymbol{B}$) has more chance of containing the *winning ticket*.

Bit-level sparsity have been considered in several areas. BSQ Yang et al. (2021) proposed a novel technique for learning the precision of mixed-precision networks by inducing the bit-label sparsity inside the sliced bits. The novel notion of bit-slice sparsity (sparsity within a sliced subdivision of binary weight) is introduced by Zhang et al. (2019) to realize efficient inference on emerging ReRAM-based DNN accelerators. Because their motivation for considering the bit-level sparsity differs from ours, their proposed method for inducing it is also totally different. Therefore, it can not efficiently sparsify the `add` in the `add-shift-add` network[8].

6 Conclusion

We propose Bit-Pruning, a novel framework for learning efficient `mult`-less sparse DNNs. The intensive evaluation shows pruning `add` (bit) of the `add-shift-add` network is more energy efficient than pruning `mult` (weight) of the `mult-add` network.

6.1 Limitations & Future work

Learning weight moving cost. In Bit-Pruning, sparse `add` for efficient inference is achieved by guiding the weights closer to the proximal weight (sparse in binary format and close to the current weight) defined in (8). The distance to current weight needs to be defined as weight moving cost $\mathcal{C}_{mv}$. In this study, we predefined it (Section 3.2). We consider this to be sub-optimal, and it might be possible to determine the cost (e.g., by learning p) from data to achieve a better tradeoff, e.g., some weight can move a lot without affecting the task loss and vice versa.

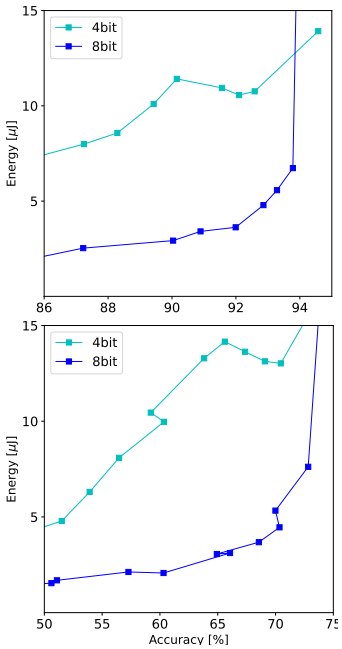


Figure 7: Ablation of bit width. CIFAR-10/100 (top/bottom).

Evaluation on quantization susceptible module. In this study, we evaluate Bit-Pruning on simple classification tasks using simple network modules such as conv. Some DNN modules are known to be more susceptible to quantization noise, such as the object detection head (Li et al., 2019a), transformer module (Bamberg et al., 2021), and recurrent module (He et al., 2016b), making it challenging to quantize them in a lower bit regime. Bit-Pruning might be an ideal choice for realizing energy-efficient inference on these modules because it could virtually utilize high-precision weight to achieve the accuracy of the original high-precision network.

Benchmark on real hardware We show the estimate of computational gain by Bit-Pruning using the basic statistic known in the literature (Table 1); we have not evaluated the actual energy consumption, wall-clock time, or area cost in the actual hardware. The proposed `add-shift-add` or existing `sparse mult-add` models require specialized hardware that supports unstructured/dynamic sparsity or optimized CPU implementation Kurtz et al. (2020) to exploit the sparsity to the full extent. The sparsity-aware accelerators often utilize in/near-memory computing architecture (Gholami et al., 2021). We believe the Bit-Pruning is more suitable than Weight-Pruning for this architecture because `add` logic is cheaper in size and energy to place in or near memory than `mult` logic.

[7]For this reason, it is difficult to compare with ShiftAddNet in terms of computational efficiency.

[8]Refer to Appendix-E,G for more discussion and experiments for comparing the ability to induce sparsity.

REFERENCES

Lennart Bamberg, Arash Pourtaherian, Luc Waeijen, Anupam Chahar, and Orlando Moreira. Synapse compression for Event-Based Convolutional-Neural-Network accelerators. December 2021.

Hanting Chen, Yunhe Wang, Chunjing Xu, Boxin Shi, Chao Xu, Qi Tian, and Chang Xu. Addernet: Do we really need multiplications in deep learning? In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 1468–1477, 2020.

Tianlong Chen, Xuxi Chen, Xiaolong Ma, Yanzhi Wang, and Zhangyang Wang. Coarsening the granularity: Towards structurally sparse lottery tickets. February 2022.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, June 2009.

Anastasia S. D. Dietrich, Frithjof Gressmann, Douglas Orr, Ivan Chelombiev, Daniel Justus, and Carlo Luschi. Towards structured dynamic sparse pre-training of BERT, 2022. URL `https://openreview.net/forum?id=-e7awdzWsOc`.

Mostafa Elhoushi. DeepShift: Implementation of "DeepShift: Towards Multiplication-Less neural networks" https://arxiv.org/abs/1905.13298.

Steven K Esser, Jeffrey L McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S Modha. Learned step size quantization. February 2019.

William Falcon and Kyunghyun Cho. A framework for contrastive self-supervised learning and designing a new approach. *arXiv preprint arXiv:2009.00104*, 2020.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. March 2018.

Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. A survey of quantization methods for efficient neural network inference. March 2021.

Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015.

Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. EIE: Efficient inference engine on compressed deep neural network. February 2016.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

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*, pp. 770–778, 2016a.

Qinyao He, He Wen, Shuchang Zhou, Yuxin Wu, Cong Yao, Xinyu Zhou, and Yuheng Zou. Effective quantization methods for recurrent neural networks. November 2016b.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

Mark Horowitz. 1.1 computing's energy problem (and what we can do about it). In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pp. 10–14, February 2014.

Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 1314–1324, 2019.

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. April 2017.

Yingyan Lin Huihong Shi, Yonggan Fu. Shiftaddnas: Hardware-inspired search for more accurate and efficient neural networks. In *Thirty-ninth International Conference on Machine Learning (ICML)*, 2022.

Yerlan Idelbayev and Miguel A Carreira-Perpinan. Low-rank compression of neural nets: Learning the rank of each layer. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2020.

Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research). a. URL `http://www.cs.toronto.edu/~kriz/cifar.html`.

Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-100 (canadian institute for advanced research). b. URL `http://www.cs.toronto.edu/~kriz/cifar.html`.

Mark Kurtz, Justin Kopinsky, Rati Gelashvili, Alexander Matveev, John Carr, Michael Goin, William Leiserson, Sage Moore, Bill Nell, Nir Shavit, and Dan Alistarh. Inducing and exploiting activation sparsity for fast inference on deep neural networks. In Hal Daumé III and Aarti Singh (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 5533–5543, Virtual, 13–18 Jul 2020. PMLR. URL `http://proceedings.mlr.press/v119/kurtz20a.html`.

Keyong Li and John Baillieul. Robust quantization for digital finite communication bandwidth (dfcb) control. *IEEE Transactions on Automatic Control*, 49(9):1573–1584, 2004.

Rundong Li, Yan Wang, Feng Liang, Hongwei Qin, Junjie Yan, and Rui Fan. Fully quantized network for object detection. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2019a.

Yuhang Li, Xin Dong, and Wei Wang. Additive Powers-of-Two quantization: An efficient non-uniform discretization for neural networks. September 2019b.

Baoyuan Liu, Min Wang, Hassan Foroosh, Marshall Tappen, and Marianna Pensky. Sparse convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 806–814, 2015.

Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11976–11986, 2022.

Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*, 2017. URL `https://openreview.net/forum?id=Skq89Scxx`.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=Bkg6RiCqY7`.

Daisuke Miyashita, Edward H Lee, and Boris Murmann. Convolutional neural networks using logarithmic data representation. March 2016.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019.

Vivek Ramanujan, Mitchell Wortsman, Aniruddha Kembhavi, Ali Farhadi, and Mohammad Rastegari. What's hidden in a randomly weighted neural network? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11893–11902, 2020.

Leslie N Smith and Nicholay Topin. Super-convergence: Very fast training of neural networks using large learning rates. In *Artificial intelligence and machine learning for multi-domain operations applications*, volume 11006, pp. 369–386. SPIE, 2019.

Ross Wightman. Pytorch image models. `https://github.com/rwightman/pytorch-image-models`, 2019.

Mitchell Wortsman, Ali Farhadi, and Mohammad Rastegari. Dnw: Discovering neural wirings (https://arxiv.org/abs/1906.00586).

Mitchell Wortsman, Ali Farhadi, and Mohammad Rastegari. Discovering neural wirings. *Advances in Neural Information Processing Systems*, 32, 2019.

Huanrui Yang, Wei Wen, and Hai Li. DeepHoyer: Learning sparser neural network with differentiable Scale-Invariant sparsity measures. September 2019.

Huanrui Yang, Lin Duan, Yiran Chen, and Hai Li. Bsq: Exploring bit-level sparsity for mixed-precision neural network quantization. *arXiv preprint arXiv:2102.10462*, 2021.

Penghang Yin, Jiancheng Lyu, Shuai Zhang, Stanley Osher, Yingyong Qi, and Jack Xin. Understanding Straight-Through estimator in training activation quantized neural nets. March 2019.

Haoran You, Xiaohan Chen, Yongan Zhang, Chaojian Li, Sicheng Li, Zihao Liu, Zhangyang Wang, and Yingyan Lin. ShiftAddNet: A Hardware-Inspired deep network. October 2020.

Jie-Fang Zhang, Ching-En Lee, Chester Liu, Yakun Sophia Shao, Stephen W Keckler, and Zhengya Zhang. SNAP: An efficient sparse neural acceleration processor for unstructured sparse deep neural network inference. *IEEE J. Solid-State Circuits*, 56(2):636–647, February 2021.

Jingyang Zhang, Huanrui Yang, Fan Chen, Yitu Wang, and Hai Li. Exploring bit-slice sparsity in deep neural networks for efficient reram-based deployment. In *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing - NeurIPS Edition (EMC2-NIPS)*, pp. 1–5, 2019. doi: 10.1109/EMC2-NIPS53020.2019.00008.

A DETAILS IN EXPERIMENT

A.1 IMPLEMENTATION

We use Pytorch (Paszke et al., 2019) for all the implementation. For experiments on CIFAR-10 and CIFAR-100, we use a single NVidia A6000 GPU. Training of 200 epochs took a few hours. For experiments on ImageNet, we use A100 DGX with four GPUs. Training of 100 epochs took five days.

A.2 NETWORK ARCHITECTURE

CIFAR-10/CIFAR-100 We use ResNet18 (He et al., 2016a) for this experiment. To adapt to the smaller input size of 32 (compared with 224 of ImageNet (Deng et al., 2009)), we follow the protocol of CIFAR-10 example used in Lighting Bolts library (Falcon & Cho, 2020); replace the first 7×7 convolution layer (stride 2) for 3×3 convolution layer (stride 1) and remove the following `max-pooling` layer. The output dimensions of the final linear layer are changed according to the dataset's number of classes (10 for CIFAR-10 and 100 for CIFAR-100).

ImageNet We use ConvNeXt-B (Liu et al., 2022). To utilize the activation sparsity, we replace `GELU` (Hendrycks & Gimpel, 2016) activation with `ReLU` activation. From their intensive ablation study (figure 2 in their paper), this change of activation function has very little impact on accuracy.

A.3 DATA AUGMENTATION

CIFAR-10/CIFAR-100 We follow the convention in the literature for training the datasets; randomly crop of 32×32 region after 4 pixel padding then applies random horizontal flip.

ImageNet We use the data augmentation techniques used in ConvNeXt (Liu et al., 2022) (settings for the fine-tuning stage) except for the stochastic depth. Specifically, we use random erasing of the probability 0.25 and label smoothing of factor 0.1. We use PyTorch Image Models library (timm) (Wightman, 2019) for the data augmentation.

A.4 DETAIL IN WEIGHT-PRUNING

For Weight-Pruning, we use magnitude-based pruning using *Hoyer-Square* Yang et al. (2019) for the magnitude computation combined with the LSQ-based quantization. In Weight-Pruning, weights elements that are zero after the quantization is pruned. The algorithm for the Weight-Pruning is similar to the one proposed in the *neural-wiring* Wortsman et al., *edge-popup* Ramanujan et al. (2020), and the ones adopted by *Hoyer-Square* Yang et al. (2019). We use quantization by LSQ instead of thresholding, and the elements of the weight that are zero after the quantization are pruned. Therefore, pruned weight can be active again, which is key to finding the pruned network's good accuracy/energy tradeoff.

We adopt the Weight-Pruning setting for a fair comparison with the proposed Bit-Pruning. In Bit-Pruning, quantized weight is converted into binary format, and elements that become zeros are pruned. In other words, both methods prune a value smaller than the threshold (e.g., half the quantization scale) where the difference between the target weight ($\mathbf{0}$ for Weight-Pruning and $\hat{\boldsymbol{W}}$ for Bit-Pruning) is evaluated using Hoyer-Square.

For Weight-Pruning, we use magnitude-based pruning using Hoyer-Square Yang et al. (2019) to compute the magnitude combined with the LSQ-based quantization. All weight elements that are zero after the quantization is pruned. We choose this setting for a fair comparison of the Weight-Pruning and Bit-Pruning. In Bit-Pruning, quantized weight is converted into binary format, and elements that become zeros are pruned. In other words, both methods prune a value smaller than the threshold (e.g., half the quantization scale) where the difference between the target weight ($\mathbf{0}$ for Weight-Pruning and $\hat{\boldsymbol{W}}$ for Bit-Pruning) is evaluated using Hoyer-Square.

A.5 DETAIL IN BIT-PRUNING

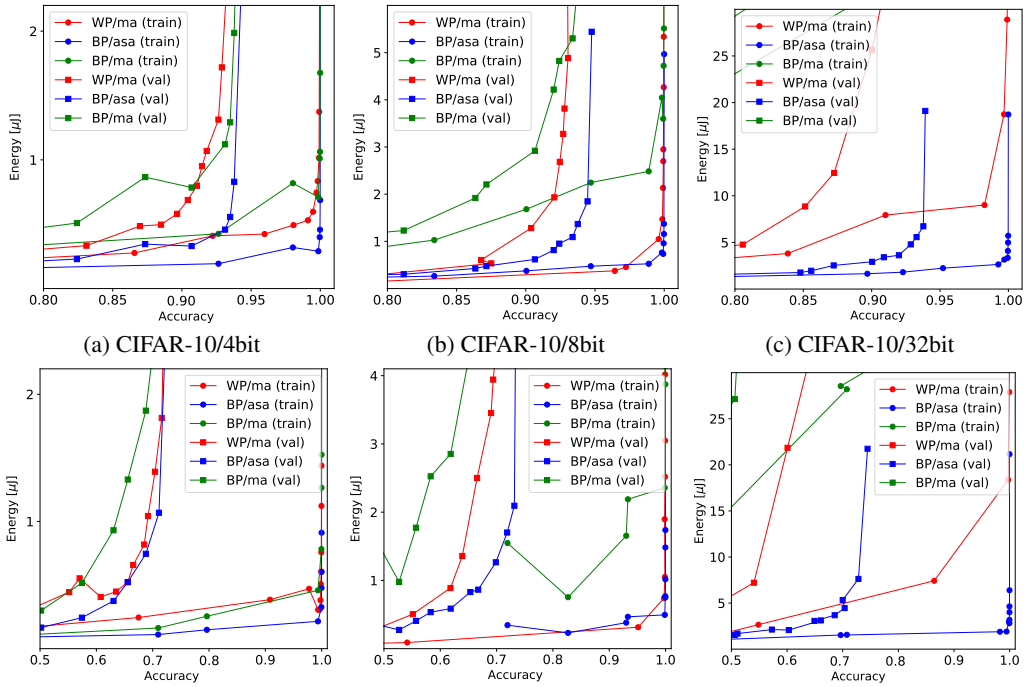
A.5.1 BALANCING PARAMETER $\lambda^{(l)}$ FOR PROXIMAL WEIGHT

As a preliminary study investigating the effectiveness of Bit-Pruning, we use the same λ for entire layers. But we can use a different value of λ for each layer. It can be set proportional to the number of operations for the given layers, or it could take the susceptibility of the weight change to the output layer into account. Learning the layer-wise λ end-to-end may also be possible using task loss. However, this requires an additional ingenuity to approximate the gradient w.r.t. λ (because of the non-differentiability f $\arg\min$ operation). In any case, it will introduce an additional hyper-parameter to be tuned. In this study, we focus on comparing the Weight-Pruning and Bit-Pruning in a simple and fair setup as possible. Therefore we left the exploration in this direction for future work.

A.5.2 SCHEDULING THE STRENGTH OF SPARSITY REGULARIZATION η

We use constant η across the entire training sequence both for Weight-Pruning and Bit-Pruning. We adopted this setting because we want to compare the Weight-Pruning and Bit-Pruning in a simple and fair configuration as possible.

Scheduling of the η will improve the accuracy/energy tradeoff; however, we prioritize the comparison on simple and fair configuration, and we left the exploration of the dynamic scheduling of η as future work.



(a) CIFAR-10/4bit (b) CIFAR-10/8bit (c) CIFAR-10/32bit

(d) CIFAR-100/4bit (e) CIFAR-100/8bit (f) CIFAR-100/32bit

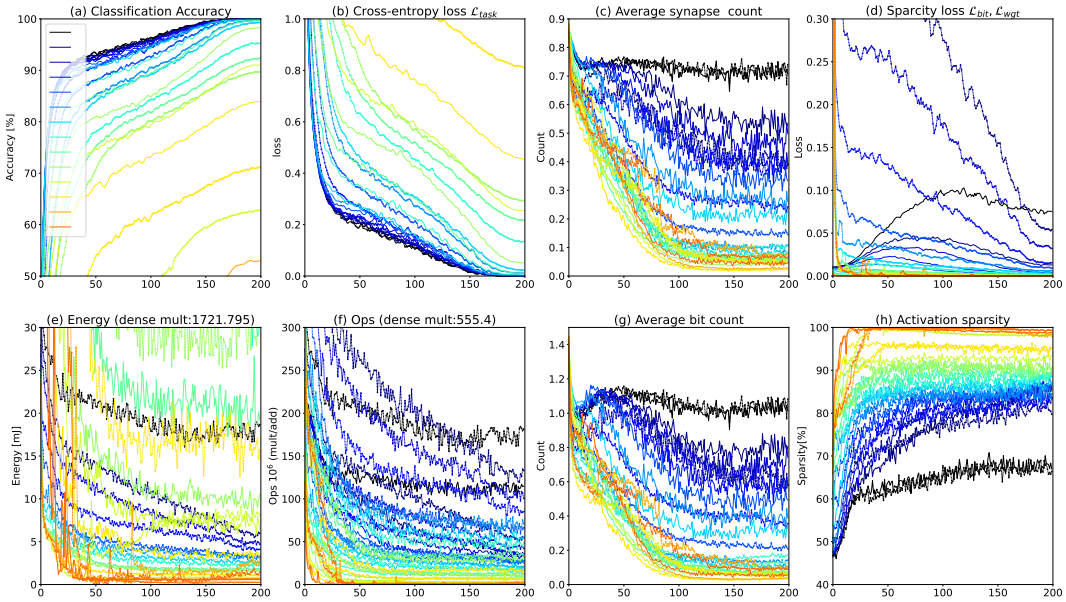
Figure 8: Accuracy/energy tradeoff of ResNet18 trained on CIFAR-10 and CIFAR-100 for difference activation quantization level. Bit-Pruning (`add-shift-add` mode) (BP/asa) and Weight-Pruning (`mult-add` mode) (WP/ma) are compared. Bit-Pruning always uses an 8bit basis, and Weight-Pruning uses the same bit for weight as activation. The result of Bit-Pruning in `mult-add` mode (BP/ma) is provided for reference.

B ADDITIONAL EXPERIMENT RESULT

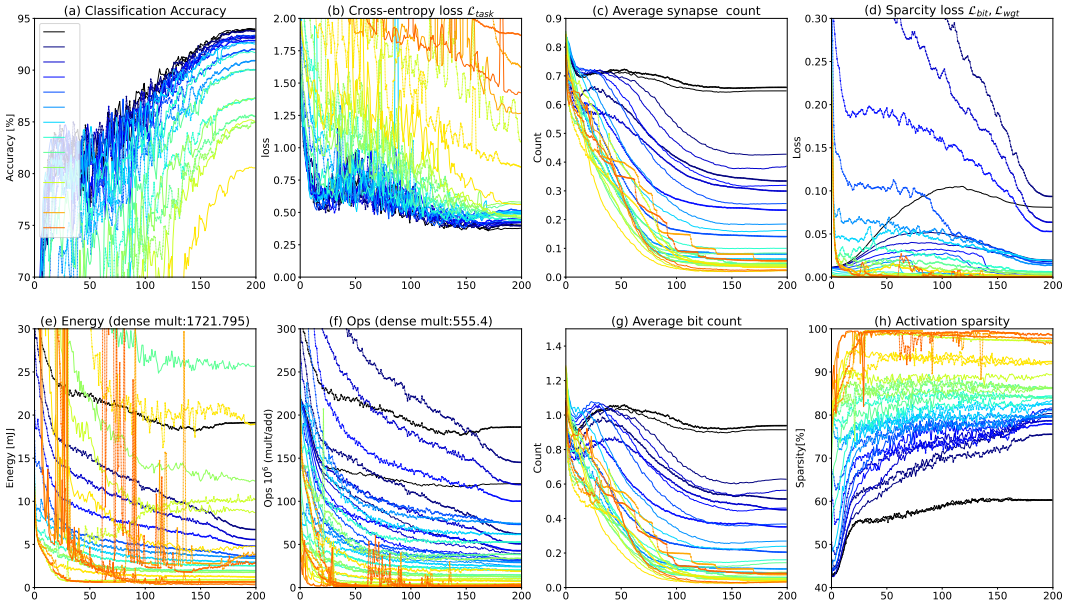
As a supplemental result of the accuracy/energy comparison experiment (Figure 4) presented in the main paper, the results, including the training accuracy, are shown in Figure 8 for CIFAR-10 (Figure 8a,8b,8c) and CIFAR-100 (Figure 8d,8e,8f). The training loss of Bit-Pruning is lower than that of Weight-Pruning for the same computational energy. It suggests that the `add-shift-add` network has more capacity than the `mult-add` network given the same energy.

Figure 9 - 10 shows the evolution of major statics during training epochs. The results presented in Figure 8 in the main paper is a summary of (a) and (e) of Figure 9 - 10.

Using a large weight for the sparsity regularization η (both for Weight-Pruning and Bit-Pruning), both weight and activation become sparse (Figure 9 - 10 (c), (g), (h)). This is because the sparsity regularization ($\mathcal{L}_{wgt}$ of (3) for Weight-Pruning and $\mathcal{L}_{bit}$ of (7) for Bit-Pruning) can be reduced not only by sparsifying the `mult` or `add` but also by suppressing the activation by promoting the activation to the minus region of `ReLU` activation function.



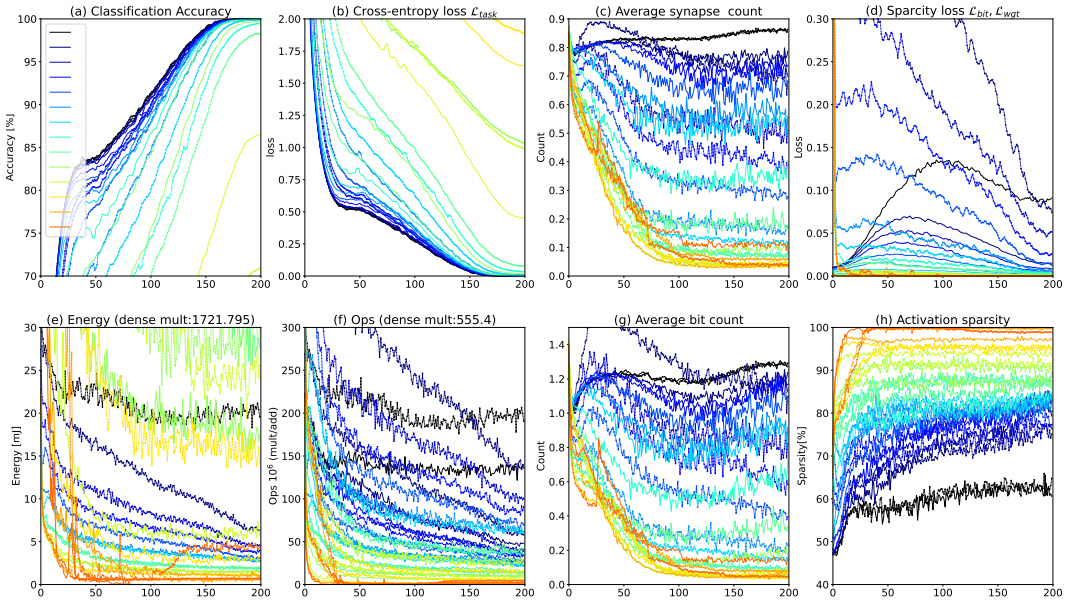
(a) Training



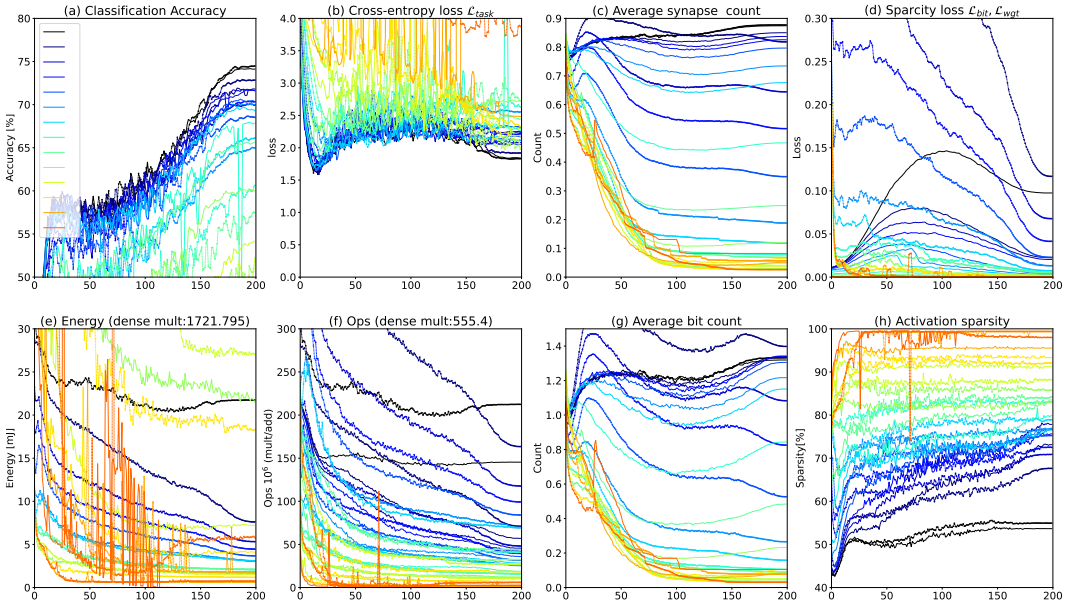
(b) Validation

Figure 9: Evolution of major statics on CIFAR-10 (Krizhevsky et al., a). Weight-Pruning (solid line) vs. Bit-Pruning (dash line with small dot). ResNet18 (He et al., 2016a). Color encode the strengths of sparsity regularization η ($\mathcal{L}_{wgt}$ for Bit-Pruning and $\mathcal{L}_{bit}$ for Bit-Pruning). Black: η is zero, Blue: η is small, Orange: η is large. (a) Classification accuracy in [%], (b) Task loss $\mathcal{L}_{task}$ (cross-entropy loss). (c) Average nonzero synapse count with respect to the original synapse connection (i.e., 1.0 is equivalent to dense connection.) (d) Loss for sparsification (`mult` for Wight-Pruning ($\mathcal{L}_{wgt}$) and `add` for Bit-Pruning ($\mathcal{L}_{bit}$)). (e) Estimated energy consumption from sparse `mult-add` and `add-shift-add` based on the statistics in Table 1. (f) Operation count of `mult` (Weight-Pruning) and `add` (Bit-Pruning). (g) Average bit count for each neuron (initial bit count is about 4 when initialized by Kaiming uniform for 8bit weight as shown in figure 6a). (h) Activation sparsity (i.e., ratio of zero element in $\boldsymbol{X}$)

.



(a) Training



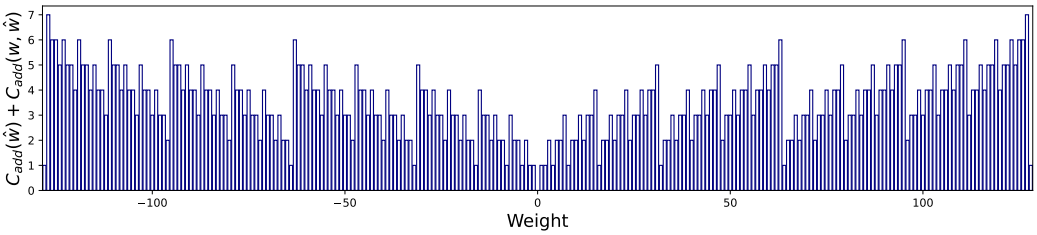
(b) Validation

Figure 10: Evolution of major statics on CIFAR-100 (Krizhevsky et al., b). Weight-Pruning (solid line) vs. Bit-Pruning (dash line with small dot). ResNet18 (He et al., 2016a). Color encode the strengths of sparsity regularization η ($\mathcal{L}_{wgt}$ for Bit-Pruning and $\mathcal{L}_{bit}$ for Bit-Pruning). Black: η is zero, Blue: η is small, Orange: η is large. (a) Classification accuracy in [%], (b) Task loss $\mathcal{L}_{task}$ (cross-entropy loss). (c) Average nonzero synapse count with respect to the original synapse connection (i.e., 1.0 is equivalent to dense connection.) (d) Loss for sparsification (`mult` for Wight-Pruning ($\mathcal{L}_{wgt}$) and `add` for Bit-Pruning ($\mathcal{L}_{bit}$)). (e) Estimated energy consumption from sparse `mult-add` and `add-shift-add` based on the statistics in Table 1. (f) Operation count of `mult` (Weight-Pruning) and `add` (Bit-Pruning). (g) Average bit count for each neuron (initial bit count is about 4 when initialized by Kaiming uniform for 8bit weight as shown in figure 6a). (h) Activation sparsity (i.e., ratio of zero element in $\boldsymbol{X}$)

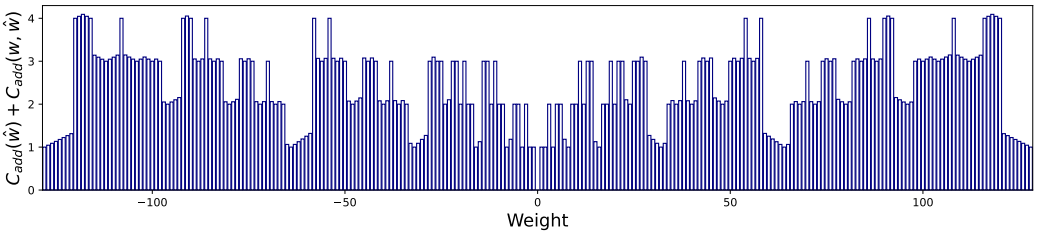
.

C LOSS LANDSCAPE OF PROXIMAL WEIGHT

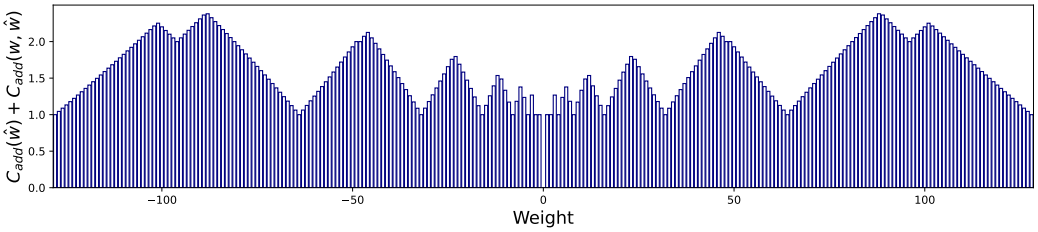
The loss landscape of the bit sparsity regularization of (7) is visualized in Figure 11. There will be multiple local minima, depending on the magnitude of $\lambda^{(l)}$. When $\lambda^{(l)}$ is very large, the local minima will be PoT and zero; on the other hand, when $\lambda^{(l)}$ is zero, it coincides with the `add` cost $\mathcal{C}_{add}$ of Figure 3a. Depending on the current weight value, the weights are guided toward the local minima except in the case when $\lambda^{(l)} = 0.0$ (In this case $\hat{\mathbf{W}}$=$\mathbf{W}$ and we won't have any gradient from the bit-sparsity regularization of (7)).



(a) Loss landscape of bit-sparsity regularization $\mathcal{L}_{bit}$ ($\lambda = 0.0$)



(b) Loss landscape of bit-sparsity regularization $\mathcal{L}_{bit}$ ($\lambda = 0.1$)



(c) Loss landscape of bit-sparsity regularization $\mathcal{L}_{bit}$ ($\lambda = 1.0$)

Figure 11: Loss landscape of *bit-sparsity regularization* $\mathcal{L}_{bit}$ ($\mathcal{C}_{mv}(w, \hat{w}) + \mathcal{C}_{add}(\hat{w})$). (b) and (c) corresponds to Figure 3b and Figure 3c in the main paper.

D MORE DISCUSSION ON COMPUTATIONAL EFFICIENCY

As discussed in the main paper (Section 3.3), we roughly get $E_{\texttt{mult}} \approx ME_{\texttt{add}}$ from Table 1 in the case of ASIC; therefore, the computational efficiency of Bit-Pruning over Weight-Pruning is almost determined by the ratio of obtained connection density $\sigma_{\texttt{add}}/\sigma_{\texttt{mult}}$.

When the network weight is initialized with a uniform random variable, then about 50% of the bits in $\mathcal{B}$ is one (Figure 6a); in this case, their computational efficiency is comparable (same order).

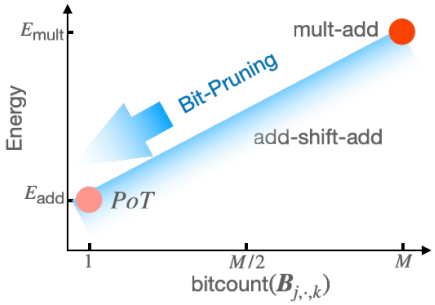


Figure 12: Fine-grained pruning of Bit-Pruning. The energy cost for the nonzero weight of `add-shift-add` ranges from $E_{\texttt{add}}$ to $E_{\texttt{mult}} \approx ME_{\texttt{add}}$. Bit-Pruning aims for $E_{\texttt{add}}$, by sparsifying $\mathcal{B}$.

By the fine-grained pruning, we expect that the average $\sigma_{\texttt{add}}$ in a Bit-Pruned network is much smaller than the average $\sigma_{\texttt{mult}}$ in a Weight-Pruned network when both networks achieve comparable accuracy (Figure 12). In the extreme case where all the nonzero weights in a network are represented as PoT, the computational cost in `add-shift-add` representation is M times smaller than that in `mult-add` representation. The `mult`-equivalent computation can be executed using roughly $E_{\texttt{add}}$ instead of $E_{\texttt{mult}}$.

E ABLATION

In this section, we compare Bit-Pruning and Weight-Pruning in a wide range of configurations that could not be converted in the main paper.

The additional results are shown only for CIFAR10 using 8-bit activation. Note that the code we'll open source includes the script to run the additional experiments on other settings (same as our main experiment result shown in Figure 4).

E.1 SMALLER SIZED NETWORK

Wight-Pruning and Bit-Pruning assume some redundancy in weights to be pruned. To investigate the behavior of both methods in a situation where the number of possible unnecessary weights is scarce, we've conducted experiments using the narrow version of ResNet18 (narrow ResNet18). The narrow ResNet18 has $8\times$ less input and output channels w.r.t the original ResNet18 we've used in the main experiments, except the input and final layer. In this situation, we can not expect significant pruning (either weights or bits).

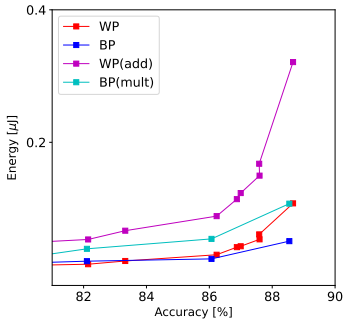


Figure 13: Ablation result from using narrow ($8\times$) network. Results using ResNet18 model with 8-bit activation on CIFAR10.

Figure 13 shows the result from using the narrow ResNet18. As expected, even when there is no sparsity regularization, e.g., $\eta = 0$, we already observe significant degeneration in accuracy compared with the original full-sized ResNet18, both on Weight-Pruning and Bit-Pruning. Increasing the weight for the regularization η, we observed the reduction in energy at the cost of accuracy degeneration in both methods. However, we observed a similar trend as in the case of the full-size network; the Bit-Pruning shows a better tradeoff than the Weight-Pruning.

E.2 DIFFERENT BALANCING PARAMETER $\lambda^{(l)}$ FOR THE PROXIMAL WEIGHT

The proximal weight $\hat{\mathcal{W}}$ is computed by finding the *best* balance between the gain from reducing the bit cost $\mathcal{C}_{bit}$ and proximity from the current weight evaluated by the weight moving cost function $\mathcal{C}_{mv}$. Intuitively, $\mathcal{C}_{add}$ encourages $\hat{\boldsymbol{W}}_{j,k}$ to be sparse in the binary format while $\mathcal{C}_{mv}$ keeps it close to the current weight $\boldsymbol{W}_{j,k}$. The *best* balance is determined by choice of λ and $\mathcal{C}_{mv}$. The proximal weight $\hat{\mathcal{W}}$ will be sparser in binary representation when λ is large.

In the main paper, we choose to use $\lambda^{(l)} = 1.0$ for all layers (Tab.3). We choose this value without careful parameter search (e.g., by evaluating the test accuracy) but by simply looking at the loss landscape of Figure 11. We consider it will induce sparsity as its local minimum is mainly composed of one nonzero element (except two nonzeroes at 96 and all zero at 0). To see the results when $\lambda^{(l)}$ is small where there are more local minimums (Figure 11-(b)), we conducted an experiment using a smaller value of $\lambda^{(l)} = 0.1$ for all layers.

Figure 14 shows the result from using smaller value of $\lambda^{(l)}$ ($\lambda^{(l)} = 0.1$). We observe the opposite results from the main experiment; the accuracy/energy tradeoff of Bit-Pruning is worse than that of Weight-Pruning.

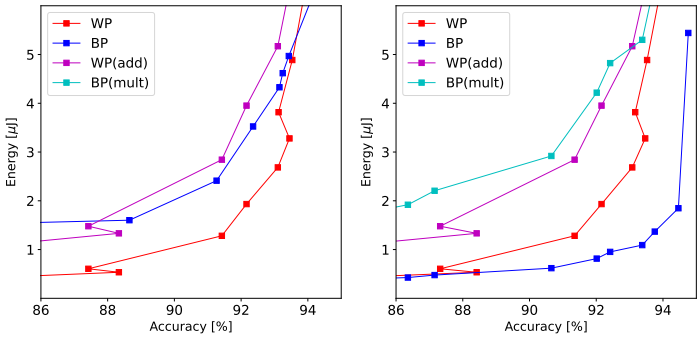


Figure 14: Ablation result from using smaller λ ($\lambda = 0.1$, left) compared with the results using larger λ ($\lambda = 1.0$) (right, same as Figure 4-(b)). Results using ResNet18 model with 8-bit activation on CIFAR10.

The degenerated tradeoff may be attributed to the too-conservative proximal weight, or it may come from the difficulty in training the weight using the loss function that has too many local minima (Figure 11-(b)).

E.3 Fine tuning vs. Training from scratch

In the main paper, we trained the ResNet18 network from scratch for CIFAR10/100 and fine-tuned the ConvNeXt network from a pre-trained network for ImageNet. We trained the network from scratch because we wanted to compare Weight-Pruning and Bit-Pruning in a fair and simple setting as possible. We were concerned the results may be affected by the pre-trained network. (We use a pre-trained network for ConvNeXt because we could not run large-scale training using of larger dataset such as ImageNet-22K.)

Nonetheless, it is important to see which performs better to consider the application scenario of Bit-Pruning (and also Weight-Pruning). For this, we conducted experiments to compare the fine-tuning and the train-from-scratch. For the pre-trained network, we used the network trained as $\eta = 0$ for both Weight-Pruning and Bit-Pruning. We employed the training procedure used in train-from-scratch.

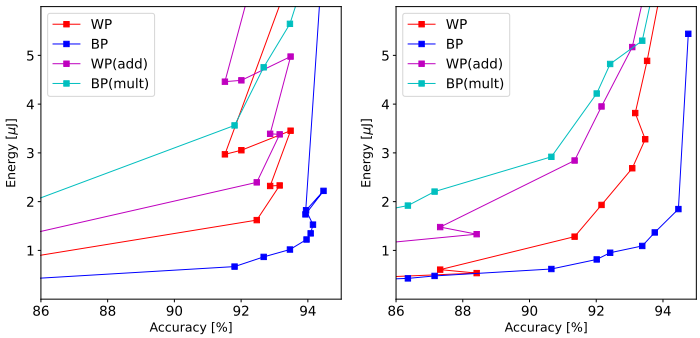


Figure 15: Ablation result of fine-tuning using pre-trained network (left) compared with the results by training from scratch (right, same as Figure 4-(b)). Results using ResNet18 model with 8-bit activation on CIFAR10.

Figure 15 compares the results from fine-tuning and train-from-scratch (same results from the Figure 4-(b) in the main paper). The results are almost the same on the high-accuracy regime. We observe the instability in the low energy regime (large η). This might be attributed to the learning-rate scheduler (e.g., warm-up, etc.) designed for the train-from-scratch scenario.

Bit-Pruning (and Weight-Pruning) could be used for both scenarios (fine-tuning and train-from-scratch). But given the similar results (at least on the high-accuracy regime), it may be beneficial to use a pre-trained network when available.

E.4 UNIFORM WEIGHT MOVING COST $\mathcal{C}_{mv}$ WITH $p = 1.0$

The proximal weight $\hat{\mathcal{W}}$ is computed by finding the *best* balance between the gain from reducing the bit cost $\mathcal{C}_{bit}$ and proximity from the current weight evaluated by the weight moving cost function $\mathcal{C}_{mv}$. Intuitively, $\mathcal{C}_{add}$ encourages $\hat{\boldsymbol{W}}_{j,k}$ to be sparse in the binary format while $\mathcal{C}_{mv}$ keeps it close to the current weight $\boldsymbol{W}_{j,k}$. The *best* balance is determined by choice of λ and $\mathcal{C}_{mv}$. In section E.2, we ablate the balancing parameter $\lambda^{(l)}$; here, we ablate the choice of the weight moving cost C_{mv}.

In the main experiments, we use $p = 1/2$ because we consider the changes in weight value when its norm is small to have more effect on the accuracy than when it is large. For example, changes of 2 when the weight was 1 (200% change) may have a more significant impact on accuracy than the weight was 200 (1%). When $p = 1/2$, C_{mv} changes a lot when its norm is small and does not change much when its norm is large.

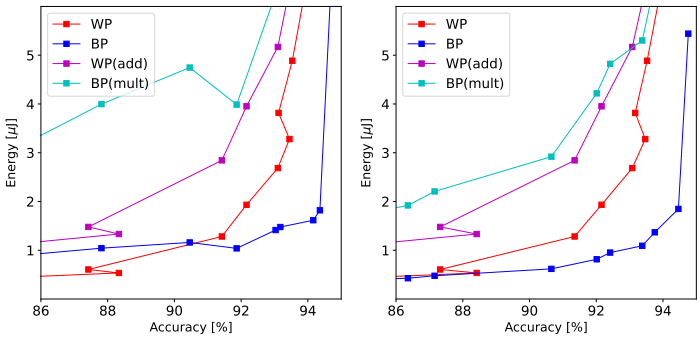
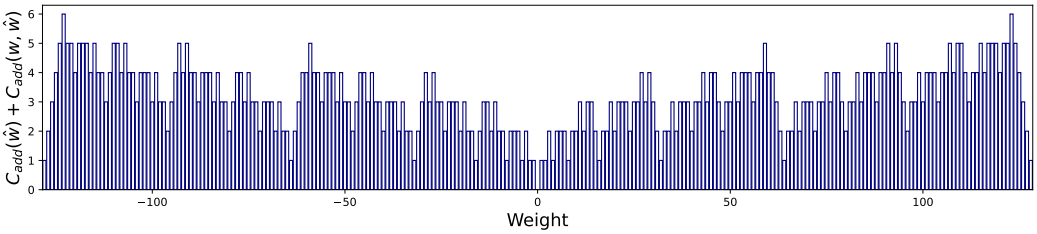


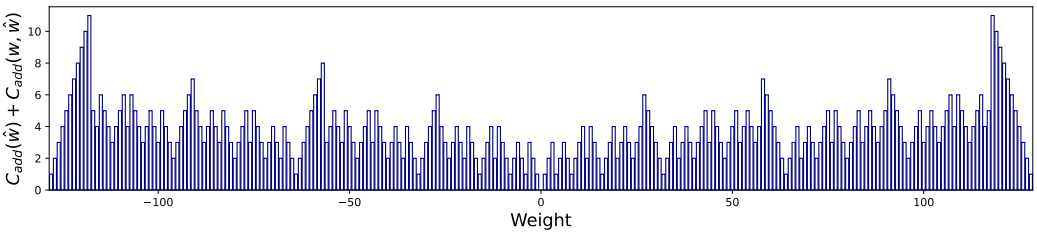
Figure 16: Ablation result of using even weight moving cost (left, $p = 1$ for $\mathcal{C}_{mv}$) compared with norm dependant weight moving cost ($\mathcal{C}_{mv}$ with) (right, same as Figure 4-(b)). Results using ResNet18 model with 8-bit activation on CIFAR10.

We conducted the ablation experiments using the moving cost C_{mv} with $p = 1.0$; this choice assumes the change in weight will equally affect the accuracy regardless of the norm of the current weight. Figure 17 visualizes the loss landscape of the bit-sparsity regularization $\mathcal{L}_{bit}$ when $p = 1.0$ which is significantly different from the case when $p = 1/2$ shown in Figure 11.

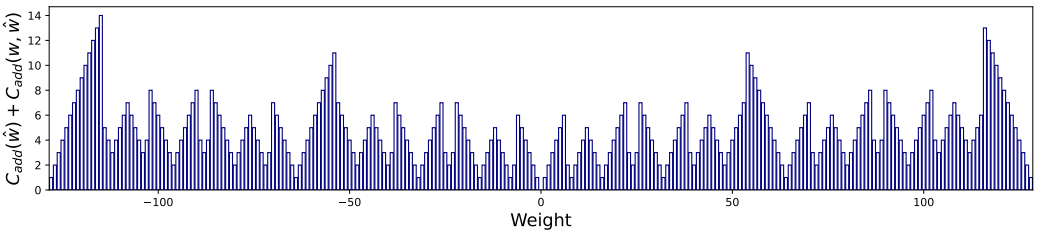
Figure 16 shows the results when $p = 1.0$. We observe a degradation in the accuracy/energy tradeoff. We suspect this is because the $\mathcal{L}_{bit}$ with $p = 1.0$ does not reflect the tradeoff of changing the value of w to the accuracy. This result suggests that we could expect a better tradeoff by using more *smart* C_{mv} e.g., by learning from data (as we discussed in future work in Sec.6).



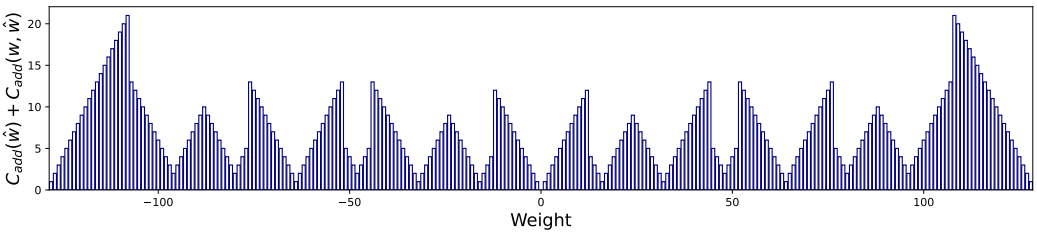
(a) Loss landscape of bit-sparsity regularization $\mathcal{L}_{bit}$ ($\lambda = 1.0$)



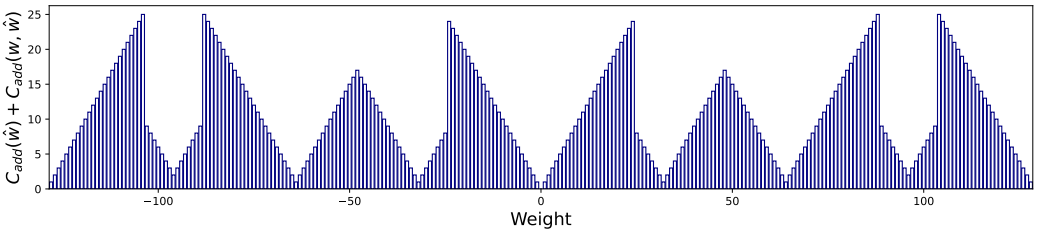
(b) Loss landscape of bit-sparsity regularization $\mathcal{L}_{bit}$ ($\lambda = 2.0$)



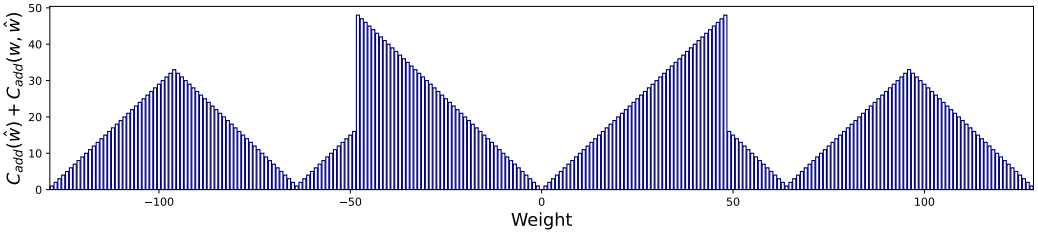
(c) Loss landscape of bit-sparsity regularization $\mathcal{L}_{bit}$ ($\lambda = 4.0$)



(d) Loss landscape of bit-sparsity regularization $\mathcal{L}_{bit}$ ($\lambda = 8.0$)



(e) Loss landscape of bit-sparsity regularization $\mathcal{L}_{bit}$ ($\lambda = 16.0$)



(f) Loss landscape of bit-sparsity regularization $\mathcal{L}_{bit}$ ($\lambda = 32.0$)

Figure 17: Loss landscape of *bit-sparsity regularization* $\mathcal{L}_{bit}$ ($\mathcal{C}_{mv}(w, \hat{w}) + \mathcal{C}_{add}(\hat{w})$) when p in $\mathcal{C}_{mv}$ is 1.0

E.5 COMPARISON WITH THE BIT-SLICE LOSS

Inducing group-wise bit sparsity (bit-slice sparsity) is explored to realize efficient DNN for emerging resistive random-access memory (ReRAM)-based DNN accelerators Zhang et al. (2019). In the ReRAM-based accelerators, each crossbar can not hold the whole 8-bit weight; therefore, a single weight element is distributed into the different crossbars, each having a 2-bit which they call a bit-sliced representation. In this hardware restriction, their goal is to sparsify each slice. The sparsity within each slice is induced by minimizing the bit-slice l_1 loss, defined as a l_1 norm of each sliced weight value (in their case, each slice has 2bit). By reducing the l_1 norm for each slice, the weight as a whole is also expected to have fewer nonzero elements in binary representation.

Ours and Zhang et al. (2019) are different in their motivation; therefore, their technique for inducing bit-sparsity is also different. Their goal is to induce the sparsity within the slice; on the hand, our goal is to induce the sparsity in `add` of the proposed `add-shift-add` formulations. They induce the sparsity by minimizing the l_1 loss within each slice and do not consider the continuity of weight as a whole; on the hand, we consider the continuity of the whole weight when inducing the sparsity to keep the change of weight value minimum to prevent the accuracy deterioration by the significant change in the weight value.

Although the motivation is different, Bit-Pruning and bit-slice sparsification of Zhang et al. (2019) share some similarities in the scene in that both aim at realizing the sparse representation of weight in binary format. Therefore, we conducted additional experiments by comparing them in terms of their ability to induce sparsity.

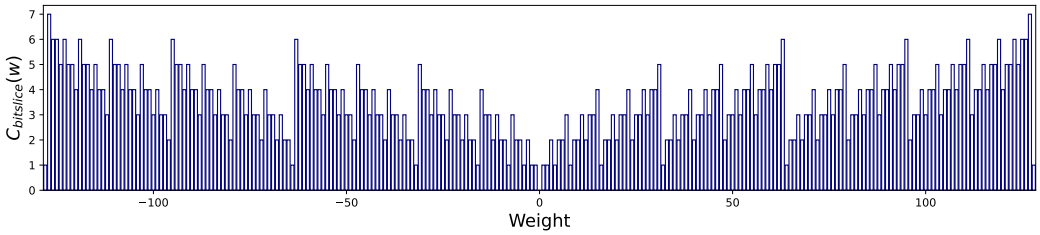
Figure 18 visualize the loss landscape of their bit-slice regularization $\mathcal{L}_{bitslice}$. Because the l_1 loss is computed inside the slice, it can not take the state of other bits into account; therefore, it can not take the continuity of the weight into account. When the number of slices is 1 (Figure 18-(d)), the bit-slice l_1 loss reduces to the $l1$ regularization for the weight value; therefore, we can not expect the sparsity as it simply reduces the norm of the weight. When the number of slices equals the bit width M (Figure 18-(a)), it guides each bit to zero, ignoring the weight value's continuity. When the number of slices is 2 (Figure 18-(c), which is their experimental setup), it only considers the continuity in the slice and ignores the continuity as weight as a whole.

For example, consider the case when $w = 63$ (6 nonzero bit), then our proximal weight is $w = 64$ (1 nonzero bit), and it is guided toward the proximal value, which differs only 1 in the weight value. Still, nonzero bits are reduced significantly (from 6 to 1). On the other hand in case of the bit-slice l_1 loss it guides the weight towards $63_{10} = (00|11|11|11)_2 \mapsto 42 = (00|10|10|10)$. The corresponding weight value change drastically, therefore, we'll expect a significant accuracy drop, and the gain for the bit sparsity is small (from 6 to 3). Our bit-sparsity loss guides the weight that maximizes the gain while preventing the weight from changing a lot by simultaneously considering the proximity of the weight value and the number of bits.

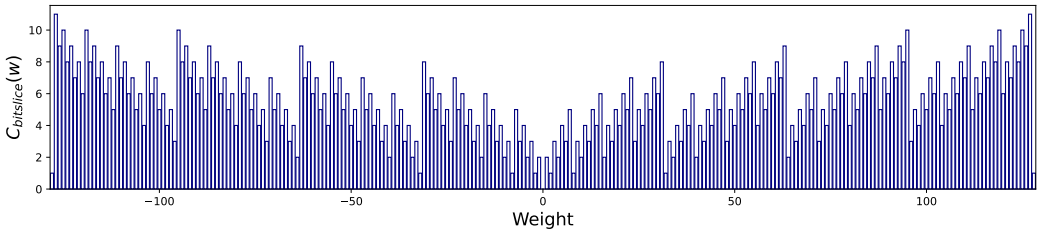
We conducted experiments to compare the result of our bit-sparsity regularization and their bit-slice l_1 regulation. We modified their loss function `calc_l1_and_zero_ratio` from their provided code[9] to be used in our Bit-Pruning framework. For a fair comparison, we also use l_1 loss (instead of Hoyer-Square loss) for our bit-sparsity regularization.

Figure 19 compare their performance. Our bit-sparsity regularization (l_1 version) shows a better tradeoff than the bit-slice l_1 regulation.

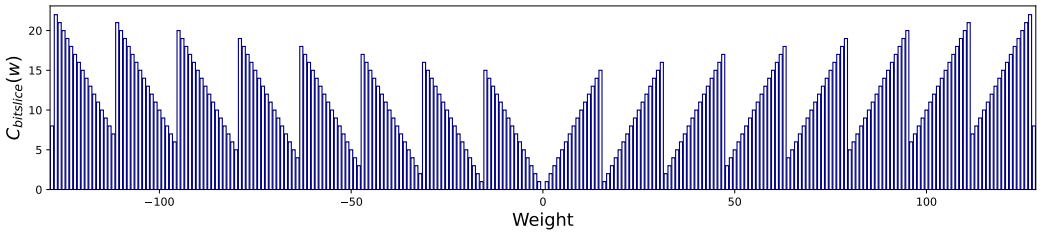
[9] `github.com/zjysteven/bitslice_sparsity`



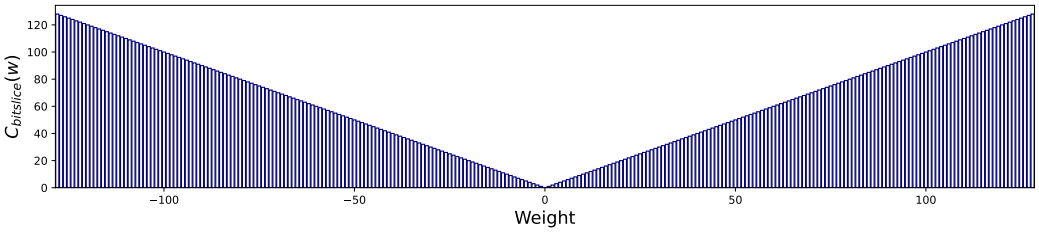
(a) Loss landscape of bit-sparsity regularization $\mathcal{L}_{bit}$ (slice=8)



(b) Loss landscape of bit-sparsity regularization $\mathcal{L}_{bit}$ (slice=4), same setting as Zhang et al. (2019)



(c) Loss landscape of bit-sparsity regularization $\mathcal{L}_{bit}$ (slice=2)



(d) Loss landscape of bit-sparsity regularization $\mathcal{L}_{bit}$ (slice=1)

Figure 18: Loss landscape of *bit-slice regularization* $\mathcal{L}_{bitslice}$ Zhang et al. (2019). When the number of slices is 1, it reduces to the $l1$ regularization. When the number of slices is a number of bit M, it simply guides each bit to zero; this is equivalent to $\lambda = 0$ in our bit-sparsity regularization.

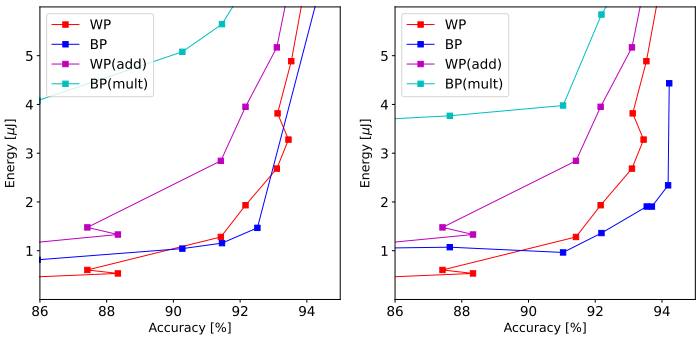


Figure 19: Ablation result using bit-slice l_1 regularization (left) compared with our bit-sparsity regularization (right, using l_1 loss instead of square-Hoyer). Results using ResNet18 model with 8bit activation on CIFAR10.

F WEIGHTED BIT-PRUNING REGULARIZATION

Our Bit-Pruning loss is defined as the difference (in Hoyer-Square sense) between the proximal weight $\hat{\boldsymbol{W}}$ and current weight $\boldsymbol{W}$. The proximal weight takes both bit-cost and proximity to the current weight into account (8). It is a compromised point (found by solving the optimization of (8)) that reduces the bit-cost while keeping the change from the current weight small as possible. The bit-cost C_{bit} of the computed proximal weight is always smaller (it could be the same) than the current weight regardless of the choice of the moving cost $\mathcal{C}_{mv}$; therefore sparsity always induces. However, the bit-sparsity regularization does NOT take the possible reduction of the bit-cost C_{bit} into account once the proximal weight is determined.

Let's consider a similar case when $w = 63$ and $w = 65$. They would have the same proximal $\hat{w} = 64$. Obviously, $w = 65$ has more nonzero bits than $w = 62$. Using the bit-sparsity regularization (E.q. 7), both $w = 63$ and $w = 65$ are guided toward $w = 64$; therefore, both weights are directed toward the sparse in binary representation having only one nonzero bit. But we'll expect more gain when $w = 63$ becomes $w = 64$ ($6 \mapsto 1$) than $w = 65$ becomes $w = 64$ ($2 \mapsto 1$). The bit-sparsity regularization of (7) does not take this into consideration for computing the gradient. Instead, they are treated equally, ignoring their different gain in bit-cost. To incorporate these gains into account, we consider using weight to reflect the gain; the weighted version of the bit sparsity regularization is defined as follows:

$$\mathcal{L}_{bit}(\mathcal{W}) = \sum_{l=1}^{L} \left| \boldsymbol{X}^{(l)} \odot (\boldsymbol{W}^{(l)} - \hat{\boldsymbol{W}}^{(l)}) \right|_0 ,$$

$$\mathcal{L}_{bit_weighted}(\mathcal{W}) = \sum_{l=1}^{L} \left| \boldsymbol{X}^{(l)} \odot (\boldsymbol{W}^{(l)} - \hat{\boldsymbol{W}}^{(l)})(C_{add}(\boldsymbol{W}^{(l)}) - C_{add}(\hat{\boldsymbol{W}}^{(l)})) \right|_0 , \tag{10}$$

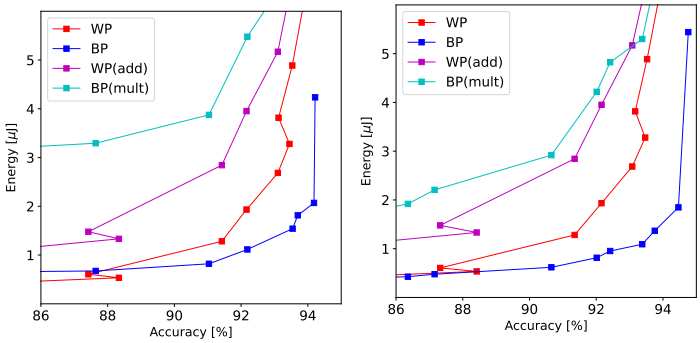


Figure 20: Ablation result of using weighted bit sparsify regularization of (10) (left) compared with the normal bit sparsify regularization on (7). Results using ResNet18 model with 8bit activation on CIFAR10.

Figure 20 shows the results. Opposed to our expectation, we could not observe a noticeable improvement in the accuracy-energy tradeoff. We have not found the reason yet; we may observe some improvement using a different optimizer, model designs, or datasets. In this work, we focused on comparing Bit-Pruning and Weight-Pruning in a simple and fair setup. We left the explanation of more advanced Bit-Pruning regularization, such as the weighting or utilization of different p for $\mathcal{C}_{mov}$ for future work.

G ADDITIONAL DISCUSSION WITH RELATED WORKS

G.1 LOW-RANK FACTORIZATION

Low-rank factorization is often utilized to reduce the number of `mult` (Howard et al., 2019) by using factorized matrix multiplication. The factorization can be realized as depthwise or 1x1 convolution; they are utilized in a computationally efficient network such as MobileNet (Howard et al., 2017) or ConvNeXt (which we evaluate in our experiment). Recent research demonstrates that the low-rank form could be learned by optimization (Idelbayev & Carreira-Perpinan, 2020). Our approach is orthogonal to this approach, and it can be combined, c.f., applying Bit-Pruning for the learnable low-rank network to further reduce the computation cost.

G.2 BSQ

BSQ Yang et al. (2021) proposed a novel method for realizing gradient-based optimization of the layer-wise precision, targeting the hardware supporting the mixed-precision. They formulate the optimization of layer-wise precision as the bit-wise optimization by reformulating the weight using the sum of the PoT basis (same as our binary representation of Eq.4). Then they directly optimize for each bit using a straight-through estimator (STE). Ours and BSQ differ in their motivation and technique for inducing bit-level sparsity. The goal of BSQ is to learn the layer-wise precision targeting the hardware supporting the dense mixed-precision operation (e.g., GPUs); on the other hand, our goal is to realize an efficient mult-free network based on the proposed add-shift-add with sparse add targeting the hardware supporting the unstructured sparsity (e.g., data-flow processors, CPUs, FPGAs, etc.). Due to the difference in their goal, BSQ applies the sparsification for *groups of weights* (e.g., each layer), while we apply the sparsification for each *individual weight element*. In BSQ, sparsity is induced by *optimizing each bit independently* using STE, which may suffer from severe quantization error. In contrast, ours *optimize in the quasi-continuous weight space* using the proximal weight by utilizing the equivalent of the add-shift-add and mult-add.