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;

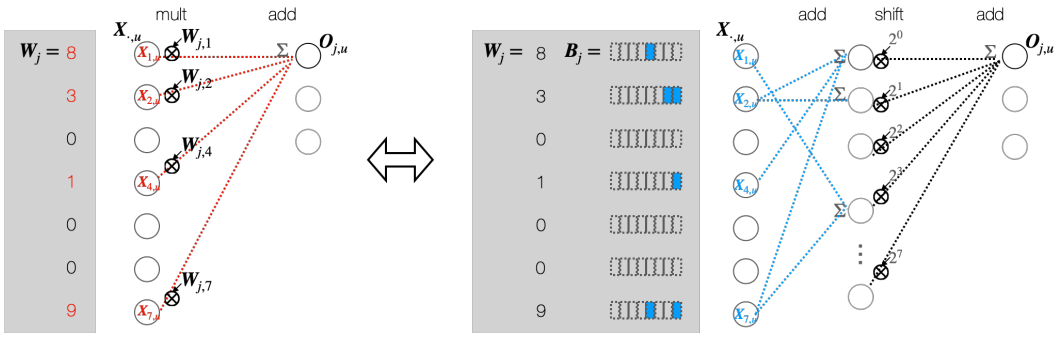


Figure 1: Dot-product realized using sparse `mult-add` (left) and equivalent representation using `add-shift-add` (in case of weight $\boldsymbol{W}_j$ is $[8, 3, 0, 1, 0, 0, 9]$). In the `add-shift-add` computation is reduced by pruning the `add`, which is realized by sparsifying the weight in binary format.

Bamberg et al., 2021; Dietrich et al., 2022; Zhang et al., 2021), realizing the efficient inference of these sparse networks.

Motivated by the significant energy reduction of sparse `mult` operations by Weight-Pruning and recent advances in the hardware supporting unstructured sparsity, we envision new frontiers in the accuracy/energy trade-off by realizing *sparse* and *`mult`-less* dot-product. 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]. Convolution with the input $\boldsymbol{I} \in \mathbb{R}^{C_i \times W \times H}$ and weight $\boldsymbol{W} \in \mathbb{R}^{C_o \times C_i \times k \times k}$ is equivalent to matrix multiplication of unfolded input $\boldsymbol{X} \in \mathbb{R}^{C_i kk \times WH}$

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

and the reshaped weight $\boldsymbol{W} \in \mathbb{R}^{C_o \times C_i kk}$:

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

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 inference of DNNs, we propose a sparse and `mult`-less dot-product by *softening* the Weight-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

Let the weights and activation be quantized to M and N bits, respectively, where M, N are sufficiently fine quantization levels such as $M, N = 8, 32$. An M-bit weight matrix[4] $\boldsymbol{W} \in \mathbb{R}^{C_o \times K}$ 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 $\boldsymbol{W}_{j,k} \in \mathbb{R}$:

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

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

[3]Some hardware only supports unstructured weight sparsity (no dynamic activation sparsity); the same argument can be made in this case by assuming the activation matrices $\boldsymbol{X}^{(l)}$ are all ones.

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

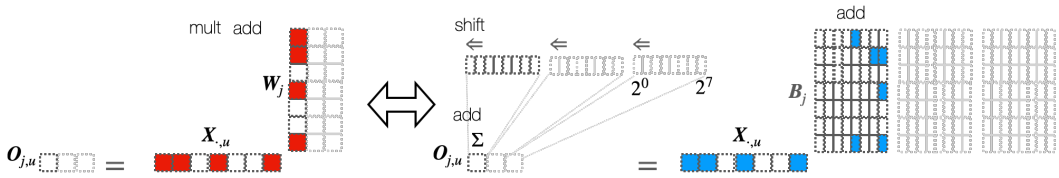


Figure 2: Matrix view of the `mult-add` (left) and `add-shift-add` (right) dot-product.

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

$$\begin{aligned} \boldsymbol{X}_{\cdot,u}^{\top}\boldsymbol{W}_j &= \sum_{k=1}^{K}\left(\boldsymbol{X}_{k,u}\cdot\sum_{m=1}^{M}2^{m-1}\cdot\boldsymbol{B}_{j,m,k}\right) \\ &= \sum_{m=1}^{M}2^{m-1}\cdot\left(\boldsymbol{X}_{\cdot,u}^{\top}\boldsymbol{B}_{j,m}\right). \end{aligned} \tag{4}$$

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 (4), 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:

$$\mathcal{L}(\mathcal{B}) = \mathcal{L}_{task}(\mathcal{B}) + \eta\ \texttt{bitcount}(\mathcal{B}), \tag{5}$$

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 $\boldsymbol{B}$, 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.

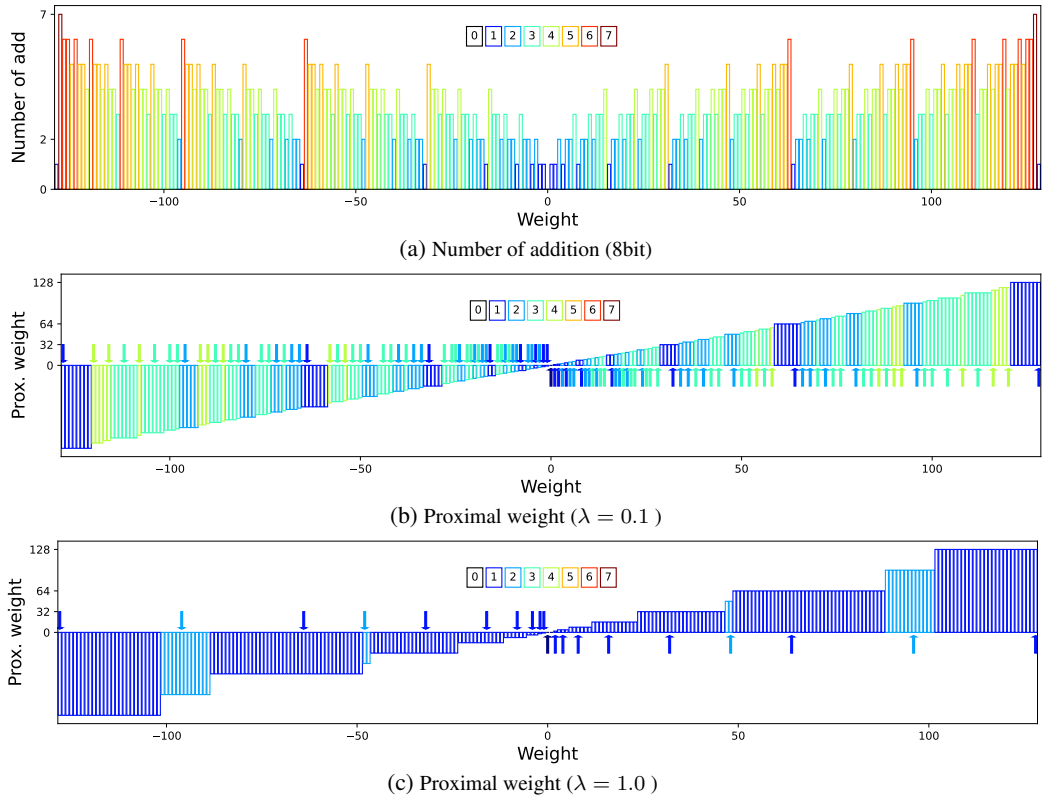
We leverage this to propose a framework called Bit-Pruning, which optimizes the network's energy consumption in a data-driven manner. The binary tensor $\boldsymbol{B}$ of the `add-shift-add` network is trained as the high-precision weight $\boldsymbol{W}$ of the `mult-add` network (Figure 2), and a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

$$\begin{aligned} \mathcal{L}(\mathcal{W}) &= \mathcal{L}_{task}(\mathcal{W}) + \eta\,\mathcal{L}_{bit}(\mathcal{W}), \\ \mathcal{L}_{bit}(\mathcal{W}) &= \sum_{l=1}^{L}\left|\boldsymbol{X}^{(l)}\odot(\boldsymbol{W}^{(l)}-\hat{\boldsymbol{W}}^{(l)})\right|_0, \end{aligned} \tag{6}$$

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 (6) 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* $\mathcal{C}_{add}$:

$$\hat{\boldsymbol{W}}_{j,k} = \arg\min_{w}\left(\lambda\,\mathcal{C}_{mv}(w,\boldsymbol{W}_{j,k}) + \mathcal{C}_{add}(w)\right), \tag{7}$$



(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 (7) (number of `add` for `add-shift-add` dot-product is encoded as color). Refer to Appendix C for the visualization of corresponding loss landscape ($\lambda\mathcal{C}_{mv}(w, \hat{w}) + \mathcal{C}_{add}(\hat{w})$).

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 $\mathcal{C}_{mv}(w, w') := |\mathrm{sgn}(w)|w|^p - \mathrm{sgn}(w')|w'|^p|$ with $p = 1/2$. 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 λ controls its balance (larger λ prefers more sparse weight). Owing to the discrete structure of the weight space (e.g., 8 bit), the solution of (7) can be analytically computed for given λ as visualized in 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. The converted network 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	$M-1$

$$\frac{E_{\texttt{add-shift-add}}}{E_{\texttt{mult-add}}} = \frac{(M\sigma_{\texttt{add}}\, K)E_{\texttt{add}} + (M-1)E_{\texttt{shift}} + (M-1)E_{\texttt{add}}}{(\sigma_{\texttt{mult}}\, K)E_{\texttt{mult}}} \approx \frac{ME_{\texttt{add}}}{E_{\texttt{mult}}}\frac{\sigma_{\texttt{add}}}{\sigma_{\texttt{mult}}} \tag{8}$$

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

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]
λ for $\mathcal{C}_{move}$ (Bit-Pruning only)	1.0 (Figure 3c)		

corresponding operations and we assume $K \gg M$ in the last approximation. In the case of ASIC, we roughly get $E_{\texttt{mult}} \approx M E_{\texttt{add}}$ 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. 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* that effectively applies different quantization levels for each weight element.

PoT dot-product. The model obtained from Bit-Pruning can be interpreted as the soft version of the (additive) PoT networks. 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 in Section 4 for visualization).

Unstructured 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. In this sense, Bit-Pruning can be seen as a soft, fine-grained version of Weight-Pruning.

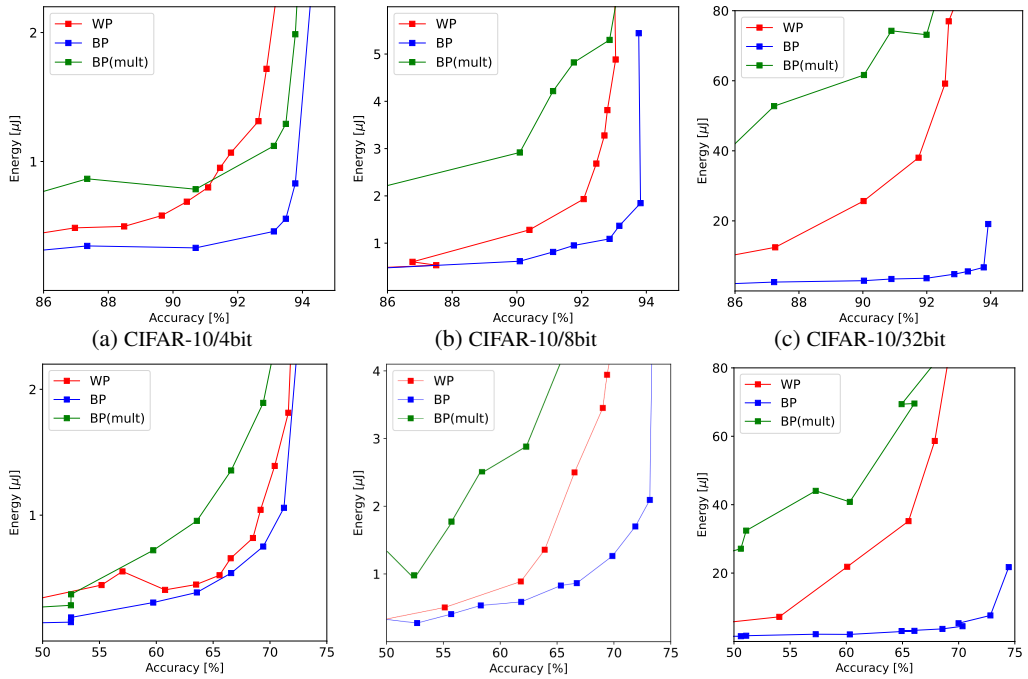
4 Experiment

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 $\mathcal{L}_{bit}$ (6) and weight-sparsity regularization $\mathcal{L}_{wgt}$ (3), we approximate l_0-norm using Hoyer-Square loss (Yang et al., 2019). We use the same network architecture and training strategy except for the loss function for both frameworks. The primary experimental setup[6] is summarized in Table 3.

[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 quantize weight using the same quantization level for activation. The result of 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. To compare the trade-off, we train networks across various sparsity coefficients η of $\mathcal{L}_{wgt}$ and $\mathcal{L}_{bit}$. Figure 4 summarizes the results on CIFAR-10 (Figure 4a,4b,4c) and CIFAR-100 (Figure 4d,4e,4f). 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.

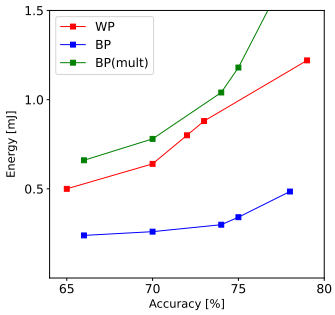


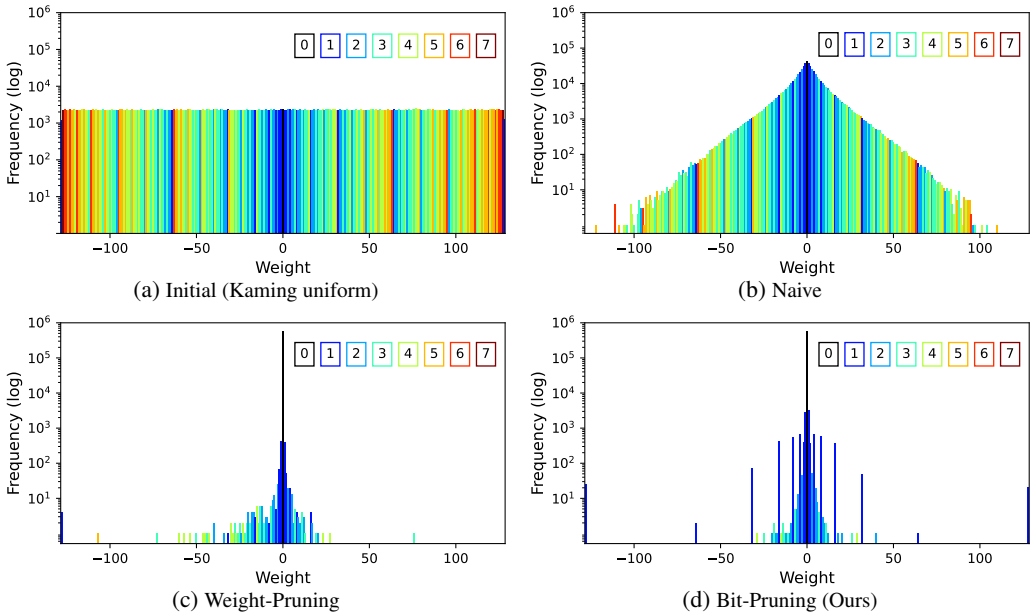
Figure 5: Result of ConvNeXt-B on ImageNet. 8bit activation. See Figure 4 for legend.

On the other hand, the Bit-Pruned network does not show efficiency in `mult-add` representation (BP(mult)). This is because the Bit-Pruned network has more nonzero elements than Weight-Pruning in terms of weight values, though it prefers sparse weights in binary format (Figure 6).

ImageNet. We conducted an experiment using 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. Bit-Pruning shows a better accuracy/energy trade-off than Weight-Pruning for large-sized, already efficient networks on a large-scale dataset.

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



(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`.

widths (4bit/8bit) (Figure 7). The result shows the network with a wide tensor (8bit) shows a better trade-off than one using the narrow tensor (4bit). This is because a network with a wide bit has more learnable parameters, and it has more choices for selecting *good* PoT values. This suggests that a better sparse network is obtained from the larger design space, as observed in 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 any particular preference for nonzero values.

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. Furthermore, they assume different types of hardware from ours; ShiftAddNet aims to realize efficient DNN training and inference for dense vector-type processors, and our target is sparse data-flow-type processors solely for inference[7].

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

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 bit width of the binary tensor $\boldsymbol{B}$) has more chance of containing the *winning ticket*.

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.

6 Conclusion

In this study, we propose Bit-Pruning, a novel framework for learning efficient `mult`-less sparse DNNs. The intensive evaluation shows that pruning `add` (i.e., bit) of the `add-shift-add` network is more accurate/energy efficient than pruning `mult` (i.e., 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 (7). 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 from data to achieve a better tradeoff, e.g., some weight can move a lot without affecting the task loss and vice versa. We left the exploitation in this direction as future work.

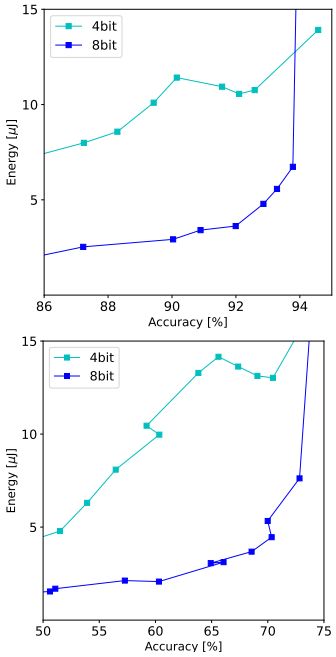


Figure 7: Results on different bit width (4bit, 8bit). CIFAR-10 (top). CIFAR-100 (bottom).

Evaluation on quantization susceptible module. In this study, we evaluate Bit-Pruning on simple classification tasks using a relatively simple network module such as convolution. 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. Evaluation of Bit-Pruning for these modules on various tasks, e.g., object detection, semantic segmentation, and optical flow estimation, would be important research.

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 to exploit the sparsity to the full extent. We have not had access to this type of hardware at the time of submission, and we leave benchmarking on real hardware for future work. They 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.

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.

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`.

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. 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.

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.

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 three days.

A.2 NETWORK ARCHITECTURE

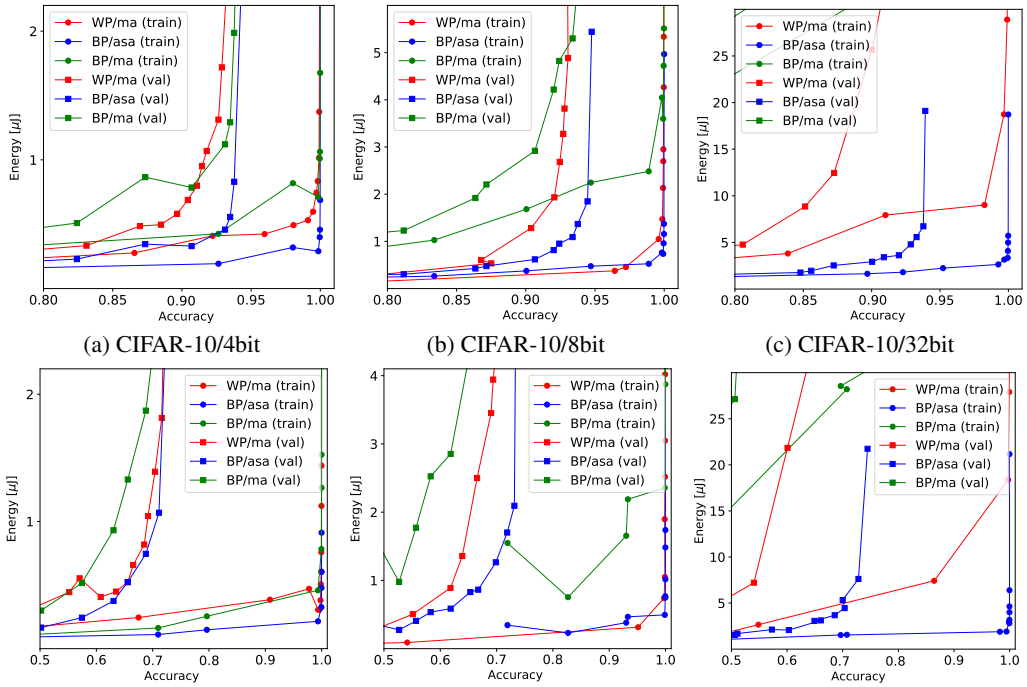
CIFAR-10/CIFAR-100 We use ResNet18 (He et al., 2016a) for this experiment. To adapt to the smaller input size of 32 (compared with 224 of ImageNet (Deng et al., 2009)), we follow the protocol of CIFAR-10 example used in Lighting Bolts library (Falcon & Cho, 2020); replace the first 7×7 convolution layer (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) CIFAR-10/4bit (b) CIFAR-10/8bit (c) CIFAR-10/32bit

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

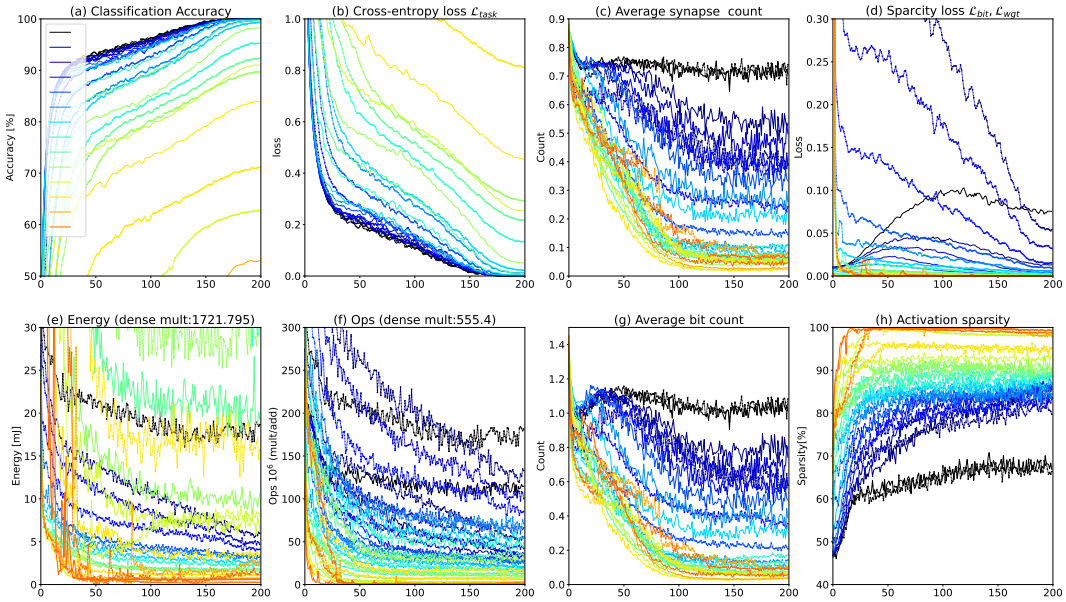
Figure 8: Accuracy/energy tradeoff of ResNet18 trained on CIFAR-10 and CIFAR-100 for difference activation quantization level. Bit-Pruning (`add-shift-add` mode) (BP/asa) and Weight-Pruning (`mult-add` mode) (WP/ma) are compared. Bit-Pruning always uses an 8bit basis, and Weight-Pruning uses the same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

B ADDITIONAL EXPERIMENT RESULT

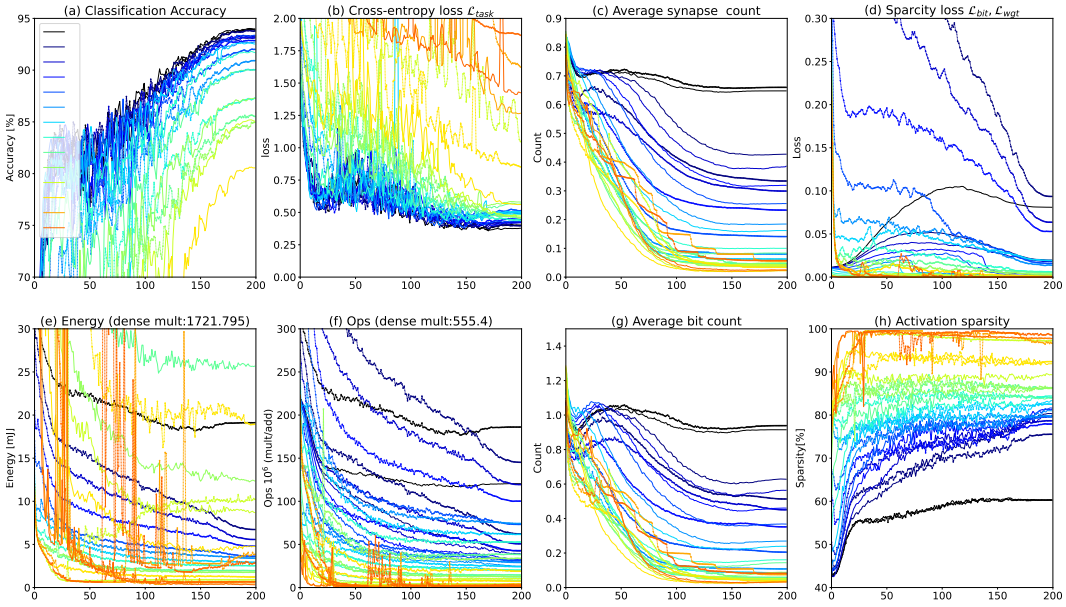
As a supplemental result of accuracy/energy comparison experiment (Figure 4) presented in the main paper, the results including the training accuracy is shown in Figure 8 for CIFAR-10 (Figure 8a,8b,8c) and CIFAR-100 (Figure 8d,8e,8f). Training loss of Bit-Pruning is lower than that of Weight-Pruning 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 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 becomes 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 (6) 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.



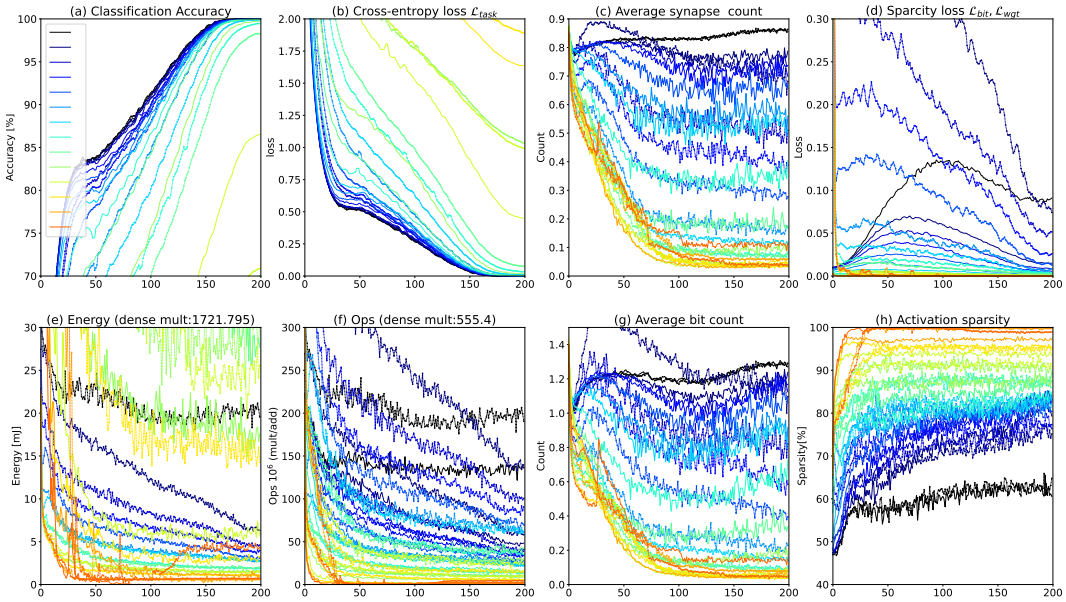
(a) Training



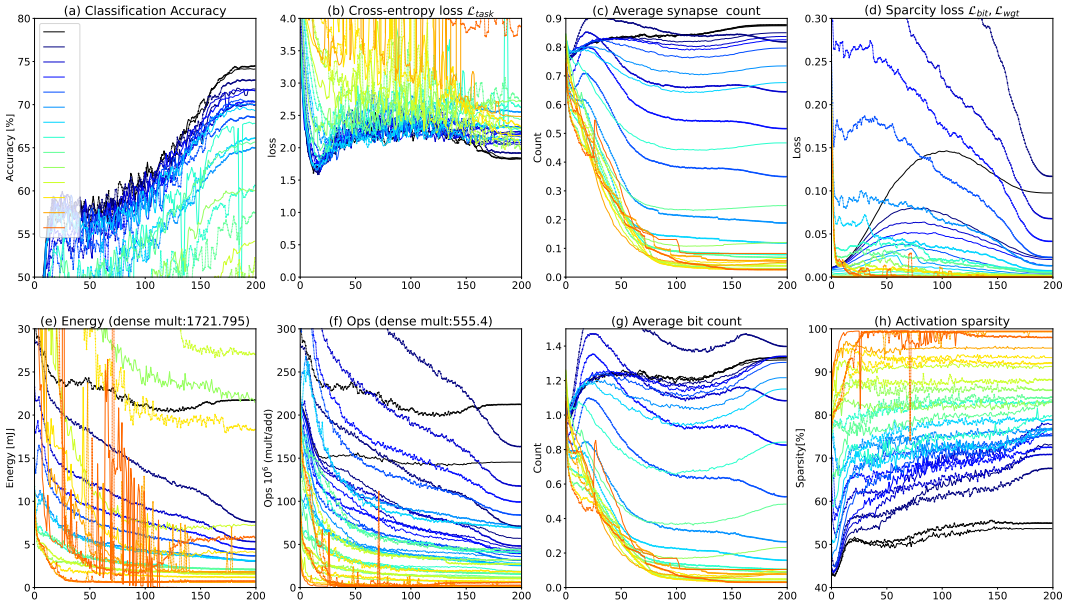
(b) Validation

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



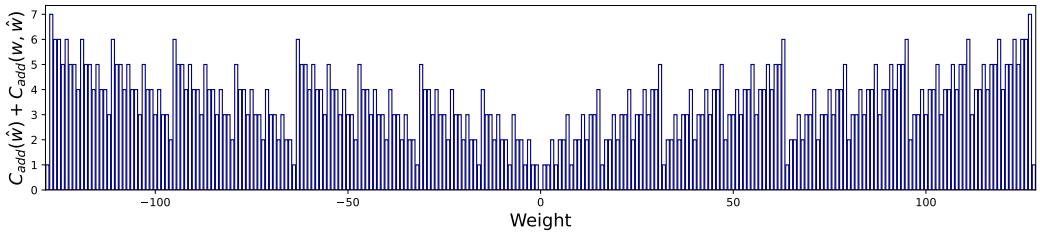
(b) Validation

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 $\boldsymbol{X}$)

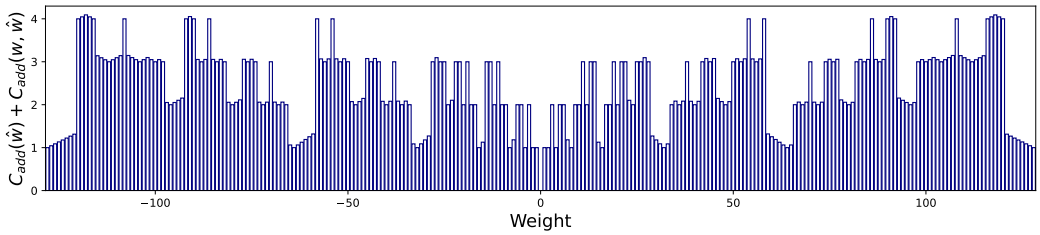
.

C LOSS LANDSCAPE OF PROXIMAL WEIGHT

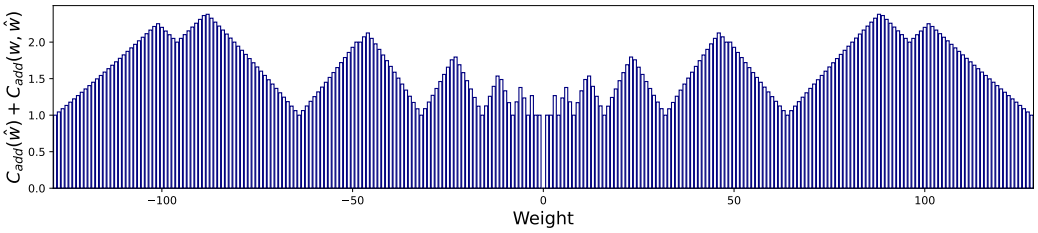
The loss landscape of the bit sparsity regularization of (6) are visualized in Figure 11. There are several local minima depending on the magnitude of λ. When λ is very large, the local minima will be PoT and zero; on the other hand, when λ is zero, it coincides with the `add` cost $\mathcal{C}_{add}$ of Figure 3a. Depending on the current weight value, the weights are guided towards the local minima.



(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* $\mathcal{L}_{bit}$ ($\lambda\mathcal{C}_{mv}(w, \hat{w}) + \mathcal{C}_{add}(\hat{w})$). (b) and (c) corresponds to Figure 3b and Figure 3c in the main paper.

D 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).

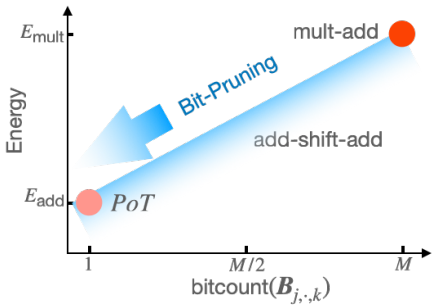


Figure 12: Fine-grained pruning of Bit-Pruning. The energy cost for the nonzero weight of `add-shift-add` ranges from $E_{\texttt{add}}$ to $E_{\texttt{mult}} \approx ME_{\texttt{add}}$. Bit-Pruning aims for $E_{\texttt{add}}$, by sparsifying $\mathcal{B}$.

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 $E_{\texttt{mult}}$.