Harmonious Coexistence of Structured Weight Pruning and Ternarization for Deep Neural Networks

Li Yang, Zhezhi He, Deliang Fan
School of Electrical, Computer and Energy Engineering
Arizona State University, Tempe, AZ 85287
{lyang166, zhezhi he, dfan}@asu.edu

Abstract

Deep convolutional neural network (DNN) has demonstrated phenomenal success and been widely used in many computer vision tasks. However, its enormous model size and high computing complexity prohibits its wide deployment into resource limited embedded system, such as FPGA and mGPU. As the two most widely adopted model compression techniques, weight pruning and quantization compress DNN model through introducing weight sparsity (i.e., forcing partial weights as zeros) and quantizing weights into limited bit-width values, respectively. Although there are works attempting to combine the weight pruning and quantization, we still observe disharmony between weight pruning and quantization, especially when more aggressive compression schemes (e.g., Structured pruning and low bit-width quantization) are used. In this work, taking FPGA as the test computing platform and Processing Elements (PE) as the basic parallel computing unit, we first propose a *PE-wise structured pruning* scheme, which introduces weight sparsification with considering of the architecture of PE. In addition, we integrate it with an *optimized weight ternarization* approach which quantizes weights into ternary values ($\{-1, 0, +1\}$), thus converting the dominant convolution operations in DNN from multiplication-and-accumulation (MAC) to addition-only, as well as compressing the original model (from 32-bit floating point to 2-bit ternary representation) by at least 16 times. Then, we investigate and solve the coexistence issue between PE-wise Structured pruning and ternarization, through proposing a *Weight Penalty Clipping (WPC)* technique with self-adapting threshold. Our experiment shows that the fusion of our proposed techniques can achieve the best state-of-the-art $\sim \mathbf{21\times}$ PE-wise structured compression rate with merely **1.74%/0.94%** (top-1/top-5) accuracy degradation of ResNet-18 on ImageNet dataset.

Introduction

In the last couple of years, the climate of Artificial Intelligence, especially deep learning, has swept various domains owing to its prominent performance over traditional methods (LeCun and others 2015). Nowadays, Deep Neural Networks (DNNs) grow into more complex structures consisting of deeper layers, larger model size, and denser connections. However, from the perspectives of hardware acceleration, DNNs still suffer from the obstacle of lightweight hardware deployment owing to the massive cost in both computation and storage. For instance, VGG-16 (Sung, Shin, and Hwang 2015) from ImageNet (ILSVRC 2014) requires 552MB parameters and 30.8 GFLOP per image, which is difficult to deploy in resource-limited mobile systems. Many recent works have been proposed to compress large DNNs, mainly including network quantization (Hubara et al. 2017), low-rank approximation (Denton and others 2014), weight pruning (Han et al. 2015) and knowledge distillation (Hinton and others 2015).

Table 1: Comparison of DNN compression techniques. Comp. and Acc. are abbreviation of Compression and Accuracy. Specific values are reported in Table 6.

	Structured sparsity	Quantized weight	Comp. rate	Acc. drop
Group-Lasso pruning	✓	✗	Low	Low
Weight ternarization	✗	✓	Moderate	Low
Naive combination	✓	✓	Moderate	Moderate
This work	✓	✓	High	Low

Weight pruning reduces model size via making partial weights as zero-value. Previous studies of weight pruning can be generally put into two categories: non-structured pruning (Han et al. 2015; Han, Mao, and Dally 2015) and structured pruning (Lebedev and Lempitsky 2016; Liu et al. 2015; Wen et al. 2016). The main difference between these two counterparts is the regularity of sparse weight pattern after pruning. The non-structured pruning attempts to generate highly irregular sparse weight pattern, targeting to achieve highest sparsity, which makes it extremely difficult to encode sparse weight efficiently due to the sparse indexing. Even though the non-structured pruning normally shows less compression accuracy degradation owing to its relatively higher pruning flexibility, its performance improvement on hardware deployment is not quite tempting. In contrast to that, the structured pruning introduces weight sparsification in a regular fashion (e.g., kernel-wise/channel-wise (Wen et al. 2016)), which makes it more hardware friendly for hardware deployment.

In addition, weights quantization is to quantize the weights of DNN into multiple discrete levels, where those quantized levels can be represented by the bit-width limited binary string. Different from weight pruning which requires crafting hardware for efficient DNN inference, quantization of weight is more straight-forward and easy to implement. In this work, we adopt the ternary value ($\{-1, 0, +1\}$) as the quantization scheme, to simplify the convolution computation complexity from MAC operations to `add/sub`-only and reduce the model size. In order to enjoy the benefits of both pruning and quantization, there are related works (Han, Mao, and Dally 2015) have shown the combination of them can achieve negligible accuracy degradation, but with moderate compression rate (e.g., 8-bit quantization bit-width) on highly redundant DNN architectures (e.g., AlexNet). Thus, those previous works fail to provide detailed insight for more aggressive compression method on the state-of-the-art DNN architectures.

In order to achieve a ternarized DNN with structured sparse weight pattern, the straightforward method is to combine the pruning and ternary quantization. However, when using the popular Group Lasso based pruning method, we found that a naive combination causes non-negligible accuracy loss. From weight pruning side, the group lasso regulation iteratively shrinks the weight distribution, meaning decrease of mean and variance of weights during training (Wen et al. 2016). However, weight ternarization prefers to loose the weight distribution (He 2019). This mismatch between weight ternarization and pruning makes network training difficult to reach optimum.

In this work, our objective is to effectively integrate the structured weight pruning and ternarization to boost the performance of DNN inference on hardware platform, with ultra-small accuracy degradation. Since we expect the improvement is maximized and valid both theoretically and applicable on practical hardware implementation, we conduct the hardware/software co-design to address various technical issues. For clarification, we summarize our main contributions in a Problem&Solution format as follows:

- **Problem 1: The structured pruning is decoupled from the hardware architecture of DNN accelerator.** Most of the previous works perform the structured pruning in the kernel-wise/channel-wise fashion (Wen et al. 2016). In our work, in order to maximize the efficiency on hardware-based DNN accelerator, we propose the processing element-wise (PE-wise) structured pruning (i.e., length of a group of weights to be pruned is identical to the capacity of PE), which is the computation core in hardware platform (i.e. FPGA, ASIC). Thus, some PEs can be easily skipped without internal modification.
- **Problem 2: There is disharmony between structured pruning and ternarization on weights of DNN, where the naive combination shows severe accuracy degradation.** For structured pruning, we adopt PE-wise group-Lasso as the weight penalty in the loss function, which tends to make entire group of weights approach to zero. For weight ternarization, the quantizer will make the close-to-zero weights tend to zero, while pushing the other weights close to positive/negative one (-1/+1) against zero. Such opposite weight evolution trend leads to a convergence dilemma. As the countermeasure, we propose the Self-Adjustable Weight Penalty Clipping (SA-WPC) which dynamically suspends penalty for weights with large absolute values. Therefore, we achieve a harmonious coexistence of structured pruning and ternarization, achieving best state-of-the-art performance. Moreover, we evaluate the performance of our proposed algorithm on both software and hardware.

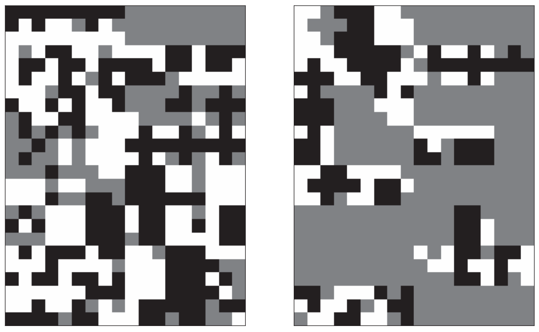


Figure 1: Stitched ternarized kernel (each kernel is in 3×3) of the first layer of ResNet-20 (He et al. 2016) on CIFAR-10, with (left) non-structured and (right) structured pruning. Patterns in {white,grey,black} denotes {-1,0,+1} respectively.

Background and Related Works

Pruning

Pruning is one of the most well-known neural network compression techniques, which enforces partial of weights be zero for both model size reduction and computation simplification (Han et al. 2015). According to the shape of weight sparsity pattern, the existing pruning technique can be generally categorized into two branches: non-structured (Han et al. 2015; Han, Mao, and Dally 2015; Srinivas and Babu 2015; Molchanov, Ashukha, and Vetrov 2017; Louizos, Welling, and Kingma 2017) and structured (Li et al. 2016; Liu et al. 2017; Lebedev and Lempitsky 2016; Wen et al. 2016; Alvarez and Salzmann 2016; He, Zhang, and Sun 2017). As the example shown in Fig. 1, structured pruning method leads to sparsity patterns (color in grey) with highly regular shapes. The division of aforementioned pruning methods are mainly hardware-oriented, where the DNN inference hardware (e.g., GPU/FPGA accelerators) can barely benefit from the random sparse pattern (Fig. 1 left) of structured pruned DNN. On the contrary, the structure pruning is a more feasible solution for hardware inference acceleration, through leveraging the regular sparsity pattern (Fig. 1 right).

Non-structured pruning. The study of non-structured pruning is originated from optimal brain damage/surgeon (LeCun, Denker, and Solla 1990; Hassibi and Stork 1993), which further explored by Han *et al.* in (Han et al. 2015; Han, Mao, and Dally 2015) on DNN. A simple pruning

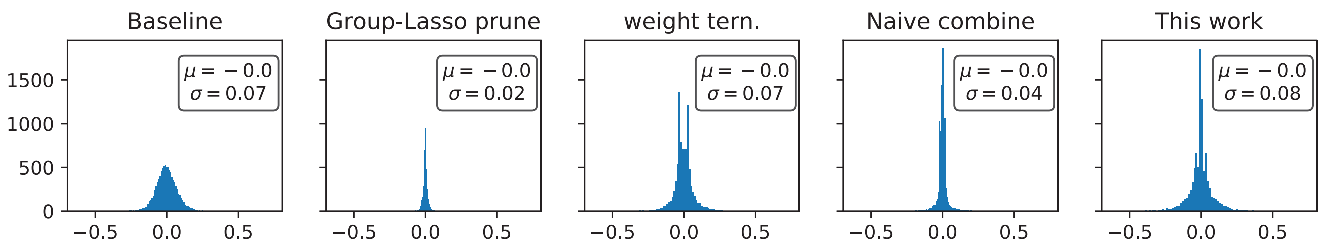


Figure 2: Sample weight distribution of ResNet-20 (He et al. 2016) layer under different training configurations, with mean (μ) and standard deviation (σ). Baseline: Full-precision model without compression; Group-Lasso prune: structured weight pruning utilizing group Lasso technique (Wen et al. 2016); Weight tern.: weight ternarization with method introduced in (He 2019); Naive combine: Naively combining the aforementioned pruning and ternarization; This work: Combining pruning and ternarization with proposed weight penalty clipping technique.

strategy is adopted in (Han et al. 2015; Han, Mao, and Dally 2015), which iteratively removes weights (i.e., convert weight to zero) with their magnitude below a pre-set threshold and fine-tune the weights for accuracy recovery. In (Srinivas and Babu 2015), Srinivas *et al.* propose a data-free pruning algorithm to remove redundant neurons iteratively by wiring similar neurons together. The variational dropout technique is utilized in (Molchanov, Ashukha, and Vetrov 2017) redundant weights pruning. In (Louizos, Welling, and Kingma 2017), DNN learns its sparse weights through the L_0-norm regularization based on stochastic gate.

Structured Pruning As the structured counterpart, various sparsity pattern in shape of channel/kernel/customized-group are explored in different works (Li et al. 2016; Liu et al. 2017; Lebedev and Lempitsky 2016; Wen et al. 2016; Alvarez and Salzmann 2016; He, Zhang, and Sun 2017). In (Li et al. 2016), the unimportant filters are directly pruned based on its L_1-norm. Liu *et al.* (Liu et al. 2017) introduce L_1 regularization on the scaling coefficient of batch normalization layers as a penalty term, where the channels with small scaling coefficient will be removed. In contrast to the aforementioned two works, all the structured pruning methods in (Lebedev and Lempitsky 2016; Wen et al. 2016; Alvarez and Salzmann 2016; He, Zhang, and Sun 2017) share the identical core technique: group Lasso.

Group Lasso is initially introduced in (Yuan and Lin 2006), then Wen *et al.* (Wen et al. 2016) append it as additional term in the loss function when training the DNN with back-propagation for learning the structured sparse weight pattern. The loss function can be formalized as:

$$\hat{\mathcal{L}} = \mathcal{L}(f(\boldsymbol{x}; \{\boldsymbol{W}_l\}_{l=1}^L), \boldsymbol{t}) + \lambda \underbrace{\sum_{l=1}^{L} \sum_{i=1}^{G_l} \overbrace{\mathcal{P}(\boldsymbol{W}_{l,i})}^{\text{Intra-group } L_2\text{-norm}}}_{\text{Inter-group } L_1\text{-norm}} \quad (1)$$

where $f(\boldsymbol{x}; \{\boldsymbol{W}_l\}_{l=1}^L)$ computes the outputs of DNN parameterized by $\{\boldsymbol{W}_l\}_{l=1}^L$ w.r.t the input $\boldsymbol{x}$. $\mathcal{L}(\cdot, \cdot)$ is the objective function of DNN (e.g., cross-entropy loss in this work). $\mathcal{P}(\boldsymbol{W}_{l,i}) = ||\boldsymbol{W}_{l,i}||_2$ calculates the Euclidean norm of the indexed weight group $\boldsymbol{W}_{l,i}$. The second term in the R.H.S of Eq. (1) is the L_1-norm of $\{\mathcal{P}(\boldsymbol{W}_{l,i})\}$ (aka. group Lasso (Yuan and Lin 2006)), which acts as the group-wise weight penalty for improving the group-wise sparsity during the optimization. G_l is the number of groups in l-th layer, and λ is the hyper-parameter to be tuned based on the dataset.

Weight Quantization

Besides pruning, weight quantization is another popular model compression technique. The main idea of weight quantization is to reduce the bit-width of weight representation format (e.g., 32-bit floating-point to 8-bit integer (Han, Mao, and Dally 2015)). A significant amount of research efforts (Leng et al. 2018; He 2019) have been invested on this topic to achieve similar goal: compressing DNN to lower bit-width (i.e., higher compression rate) while minimizing the accuracy degradation in comparison to its full-precision baseline.

In this work, we follow the weight ternarization method introduced in (He 2019) as low bit-width quantization baseline, which is briefly introduced as follow. Given the weights $\boldsymbol{W}_l$ in l-th layer of DNN, the weight ternarization function can be expressed as:

$$\hat{w}_{l,i} = \text{Tern}(w_{l,i}, \Delta_l) = \alpha_l \cdot \begin{cases} +1 & w_{l,i} > \Delta_l \\ 0 & -\Delta_l \leq w_{l,i} \leq \Delta_l \\ -1 & w_{l,i} < -\Delta_l \end{cases} \quad (2)$$

$$\alpha_l = \mathbb{E}(|\{w_{l,i}\}|), \quad \forall \{i | |\boldsymbol{W}_{l,i}| > \Delta_l\} \quad (3)$$

where $\hat{w}_{l,i}$ is the i-th ternarized weight element in l-th layer. α_l is the layer-wise scaling coefficient (i.e., quantized value). The threshold $\Delta_l = 0.05 \cdot \max(|\boldsymbol{W}_l|)$. As other quantization work, the Straight-Through Estimator (STE) (Bengio, Léonard, and Courville 2013) method is used as the countermeasure against non-differential issue of staircase quantization function (Eq. (2)). Thus, with the given vectorized input $\boldsymbol{x}$ and target $\boldsymbol{t}$, the optimization of ternarized DNN can be written as

$$\begin{aligned} \min_{\{\boldsymbol{W}_l\}_{l=1}^L} \quad & \mathcal{L}(f(\boldsymbol{x}; \{\hat{\boldsymbol{W}}_l\}_{l=1}^L), \boldsymbol{t}) \\ s.t. \ \{\hat{\boldsymbol{W}}\}_{l=1}^L &= \text{Tern}(\{\boldsymbol{W}\}_{l=1}^L) \end{aligned} \quad (4)$$

where this equation uses the identical notations as in Eq. (1).

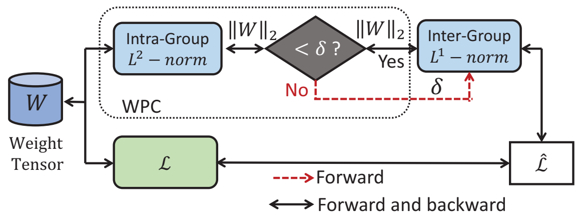


Figure 3: The overview of weight penalty clipping with self-adapting threshold.

Methodology

Problem formalization

As described in the related work section, both group-lasso based structured pruning and weight ternarization show great performance in compressing the model while maintaining the inference accuracy w.r.t the full-precision baseline. For maximizing the model compression rate and enjoying the benefits from both methods, we first naively combine them together, where the DNN training process can be described as minimizing the following loss function:

$$\hat{\mathcal{L}} = \mathcal{L}(f(\boldsymbol{x}; \mathrm{Tern}\{\boldsymbol{W}_l\}_{l=1}^{L}), \boldsymbol{t}) + \lambda \sum_{l=1}^{L}\sum_{i=1}^{G_l} \mathcal{P}(\boldsymbol{W}_{l,i}) \quad (5)$$

Nevertheless, such naive combination leads to severe accuracy degradation without significantly improving the compression rate.

In order to identify the problem, we examine the weight distribution of different training configurations, where the corresponding histograms are depicted in Fig. 2. The results show that the group Lasso based structure pruning still acts like a normal regularization term which forces the entire weights toward smaller values (i.e., small standard deviation σ with zero mean μ), in comparison to the full-precision baseline. On the contrary, weight ternarization makes negligible change on σ w.r.t the baseline. We also observe that, naively combining the structured pruning and ternarization method leads the σ converge to some value between Group-Lasso and weight ternarization counterparts. Such observation inspires us about the potential cause of accuracy degradation when naively combining two methods:

- Is that applying the group Lasso (i.e., weight penalty) upon entire weights contradictory to the weight ternarization method?
- Whether maintaining the weight distribution close to the original ternary counterpart helpful to mitigate the accuracy degradation?

Based on those inspirations, we propose a *Weight Penalty Clipping* (WPC) technique with self-adapting threshold as the countermeasure to address the coexistence issue between group Lasso based structure pruning and ternarization, which is specified in the following subsection.

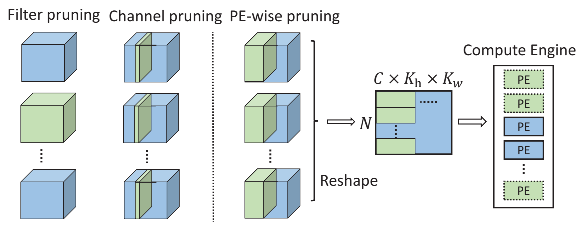


Figure 4: PE-wise pruning

Weight Penalty Clipping with self-adapting threshold

To address above discussed issue, further adjustment is made on the intra-group L_2-norm term in Eq. (1), thus the Eq. (5) can be rewritten as:

$$\hat{\mathcal{L}} = \mathcal{L}(f(\boldsymbol{x}; \mathrm{Tern}\{\boldsymbol{W}_l\}_{l=1}^{L}), \boldsymbol{t}) + \lambda \sum_{l=1}^{L}\sum_{i=1}^{G_l} \underbrace{\min\big(||\boldsymbol{W}_{l,i}||_2; \delta_l\big)}_{\text{Weight Penalty Clipping}} \quad (6)$$

$$\delta_l = a \cdot \frac{1}{G_l}\sum_{i=1}^{G_l} ||\boldsymbol{W}_{l,i}||_2 \quad (7)$$

where δ_l denotes the layer-wise self-adapting clipping threshold, which is utilized to mitigate the intra-group L_2-norm penalty on large weights, and a is scaling coefficient. Note that, once the intra-group L_2-norm penalty of $\boldsymbol{W}_{l,i}$ is clipped, the inter-group L_1-norm penalty is clipped as well. We name such method as the weight penalty clipping. Fig. 3 shows the processing of WPC. In each training iteration, the updated weights are used to the loss function as shown in Eq. (7). Then after intra-group L_2-norm calculation, by comparing with a threshold δ_l, WPC will decide whether the corresponding $||\boldsymbol{W}_{l,i}||_2$ will be used on loss function and go backward. Considering two cases:

- When $||\boldsymbol{W}_{l,i}||_2 \geq \delta_l$, it indicates that weights in $\boldsymbol{W}_{l,i}$ are relatively large (i.e., important) which are not supposed to be pruned by the group-Lasso term in 6. Then, the weight penalty clipping is performed which replaces the weight penalty of $||\boldsymbol{W}_{l,i}||_2$ in $\hat{\mathcal{L}}$ with δ_l. Hereby, we have to highlight that δ_l is treated as a constant, where its calculation is removed from the backward computation graph.
- When $||\boldsymbol{W}_{l,i}||_2 < \delta_l$, we keep the weight penalty of $||\boldsymbol{W}_{l,i}||_2$ in its original value, thus the group Lasso term can continuously affect $\boldsymbol{W}_{l,i}$ and prune the weights in group-wise fashion.

Our simulation in the later sections will show that the proposed WPC with self-adapting threshold is helpful to mitigate the disharmony between structured pruning (i.e., group Lasso) and weight ternarization.

PE-wise structured pruning

The performance (e.g., accuracy) of DNN pruned by group Lasso based method highly relies on the defined group shape, where various group shape (e.g., channel/filter/depth/etc.) has been explored in (Wen et al. 2016). Normally, using large group capacity (i.e., number of weights per group) leads to either low group sparsity or low inference accuracy. Meanwhile, using small group capacity will be beneficial to both group sparsity and accuracy, however the pruned DNN may hardly be accelerated by targeted hardware. For balancing the DNN performance after structured pruning and inference efficiency on targeted hardware, we propose to make the group shape identical to the Processing Element (PE) of target hardware. PE is defined as the basic computing unit in modern DNN accelerator (FPGA, ASIC or GPU). Through proper hardware design, DNN accelerator can efficiently utilize the PE-wise sparsity pattern, and further speed up is observed by our later experiments.

Due to the limited hardware resource, fully paralleled computation can hardly be achieved. For PE-wise structured pruning, we define the group size equals to the capacity of one PE. In a standard convolution layer, weights are stored in a 4-D tensor $\boldsymbol{W} \in \mathbb{R}^{N \times C \times K_{\text{h}} \times K_{\text{w}}}$, where N, C, K_{h}, K_{w} denotes the output channel, input channel, kernel height and width in current layer, respectively. As shown in Fig. 4, the size of the PE-wise pruning is between the filter ($C \times K_{\text{h}} \times K_{\text{w}}$) pruning and channel ($N$) pruning. In addition, in the terms of hardware deployment, the high dimensional convolution operation is often reduced to matrix multiplication, which reshapes the 4D weight tensor to 2D matrix with the size of $(N, C \times K_{\text{h}} \times K_{\text{w}})$. Thus, defining structured sparsity groups in channel-wise (C) and shape-wise ($K_{\text{h}} \times K_{\text{w}}$) does not match well with the practical hardware implementation. We choose the capacity of PE as the group to perform group-wise pruning, with size of $C_{\text{g}} \times K_{\text{h}} \times K_{\text{w}}$. C_{g} can be divided by C. In this case, one PE could do computation between one group of weight and corresponding partial feature maps vector with size of $C_{\text{g}} \times H \times W$.

Experiment and Discussion

Experiment setup

Dataset and network structure In this work, we take the classic image classification task as example to examine the performance of our proposed technique. Two dataset are used in this work, which are CIFAR-10 (Krizhevsky and Hinton 2009) and ImageNet (Deng et al. 2009). CIFAR-10 contains 50K training samples and 10K test samples with 32×32 image size. For CIFAR-10, we adopt the ResNet-20/32/44/56 (He et al. 2016). We train the network using momentum SGD optimizer, where the initial learning rate is 0.1, which scaled by 0.1 at epoch 80 and 120 respectively. The data argumentation is identical as the configuration adopted in (He et al. 2016).

In addition, ImageNet contains 1.2 million training images and 50 thousands validation images, which are labeled with 1000 categories. For the data pre-processing, we choose the scheme adopted by ResNet (He et al. 2016). We adopt the ResNet-18 and AlexNet. We train the network using Adam optimizer, where the initial learning rate is 0.0001, which scaled by 0.2 at epoch 30, 40, 45 respectively. Similar with other quantization works, such as (Lin, Zhao, and Pan 2017; He 2019; Rastegari and others 2016), we keep the first and last layer in full precision(i.e. 32-bit float).

Table 2: Inference accuracy (%) of ResNets on CIFAR-10

	ResNet-20	ResNet-32	ResNet-44	ResNet-56
FP	91.7	92.36	92.47	92.68
Ours	90.89	91.62	91.76	92.05
Gap	-0.81	-0.74	-0.71	-0.63

Experiment platform Our algorithm is conducted on the Pytorch deep learning platform with 4-way NVIDIA Titan XP GPUs. Moreover, in order to examine the performance of PE-wise structured pruning, we properly design a FPGA-based DNN accelerator. The FPGA platform is Xilinx PYNQ-Z1 board supported by PYNQ open-source framework. It uses a Xilinx Zynq-7000 SoC containing an XC7Z020 FPGA alongside an ARM Cortex-A9 embedded processor.

Compression rate definition To fully utilize structured pruning in hardware perspective, we could use a binary indexer to index which PE group are all zeros values. By doing this, we just need to store the particular weight groups that have non-zeros values in memory. Also, the size of the binary indexer is negligible comparing with the size of weights. Thus, we formulate the compression rate of weight C as:

$$C = \frac{32}{(1 - G_s) \cdot n} \tag{8}$$

where G_s means the group sparsity which is the fraction of number of groups with all zero values over layers and n is the bit-width of model which should be 2 in here since we encode weights in binary format. 32 represents the bit-width of the full precision. It is worthy noting that for calculating the compression rate, we only consider group sparsity, since this matrix that we can real implementation in the hardware. In addition, all the compression rate of our method shown below are defined by the Eq. (8).

Results

CIFAR-10 Experiment The proposed method is tested on ResNet-20/32/44/56. The size of PE group is set to be $16 \times 3 \times 3$ for all layers of all kinds of networks. From the result of inference accuracy which are listed in Table 2, it can be seen that all these four kinds of ResNet models achieves less than 1% accuracy loss comparing with floating point baseline. Also, it shows that a more compact neural network, like ResNet-20, is more likely to encounter accuracy loss, owing to the network capacity is hampered by this aggressive model compression.

ImageNet Experiment Our proposed method is evaluated on ResNet18 and AlexNet. The size of PE group is setted to be $64 \times 3 \times 3$ for all convolutional layers.
AlexNet. To better show the effectiveness of our method, besides comparing with non-structured ternarization method as mentioned above, we also compare with related structured pruning work (Wen et al. 2016). However, the enhancement

Table 3: Simulation result of structured pruning for AlexNet (Krizhevsky, Sutskever, and Hinton 2012) on ImageNet dataset. FP. indicates full-precision (32-bit floating-point) and Tern. indicates ternary weights. Note that, Acc. and Comp. are abbreviation of accuracy and compression.

Method	Weight format	Top-1 Acc. gap (%)	Sparsity metrics	Layer index						Comp. rate
				Conv1	Conv2	Conv3	Conv4	Conv5	Conv2-5	
SSL (Wen et al. 2016)	Floating-point (32-bit)	2.06	Column (%)	0.0	63.2	76.9	84.7	80.7	-	$\sim 6.4\times$
			Row (%)	9.4	12.9	40.6	46.9	0.36		
			Layer (%)	9.4	68.8	86.3	91.9	80.8	84.4	
This work	Ternary (2-bit)	3.23	Group (%)	24.3	36.5	47.3	45.0	24.2	35.46	$\sim 24.7\times$
			In-group (%)	13.2	59.5	48.6	41.7	36.1		
			Layer (%)	34.3	80.2	72.9	67.9	51.6	68.2	

Table 4: Inference accuracy ($\%$) of ResNet18 on ImageNet

	Quan scheme	First layer	Last layer	Accuracy (top1/top/5)	Comp. rate
Baseline	-	FP	FP	69.75/89.07	$1\times$
BWN	Bin.	FP	FP	60.8/83.0	$\sim 32\times$
ABC-Net	Bin.	FP	FP	68.3/87.9	$\sim 6.4\times$
ADMM	Bin.	FP	FP	64.8/86.2	$\sim 32\times$
TWN	Tern.	FP	FP	61.8/84.2	$\sim 16\times$
TTN	Tern.	FP	FP	66.6/87.2	$\sim 16\times$
ADMM	Tern.	FP	FP	67.0/87.5	$\sim 16\times$
(He 2019)	Tern.	FP	FP	67.95/88.0	$\sim 16\times$
Ours	Tern.	FP	FP	**68.01/88.13**	$\sim 21.3\times$

Table 5: Ablation study on CIFAR-10

	Tern	Pruning	Naive combine	Ours
ResNet-20	91.62	91.1	90.01	90.89
Overall sparsity	49	61	70	50
Group sparsity	-	44	18	25
Comp. rate	$\sim 16\times$	~ 1.78	$\sim 19.5\times$	~ 19.5
ResNet-32	92.48	91.88	90.68	91.64
Overall sparsity	48	40	74	58
Group sparsity	-	34	28	43
Comp. rate	$\sim 16\times$	~ 1.5	$\sim 22.2\times$	~ 28.1
ResNet-44	92.71	92.29	91.15	91.98
Overall sparsity	55	58	80	64
Group sparsity	-	46	21	44
Comp. rate	$\sim 16\times$	~ 1.85	$\sim 20.2\times$	~ 28.6
ResNet-56	93.1	92.86	92.01	92.85
Overall sparsity	52	67	82	67
Group sparsity	-	28	42	55
Comp. rate	$\sim 16\times$	~ 1.39	$\sim 27.6\times$	~ 35.6

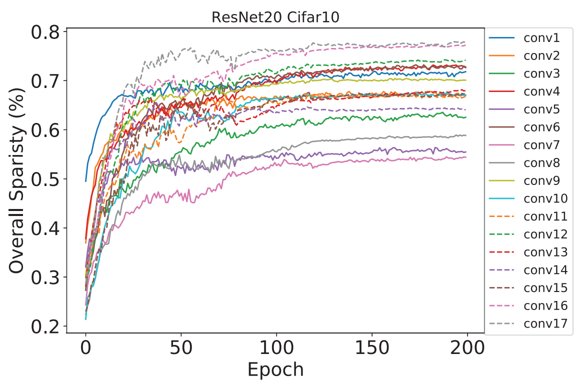
of accuracy of vanilla AlexNet causes these two works have different baselines (i.e., 61.78%, 57.4%). Thus, to conduct a relatively fair comparison, we indicate the accuracy loss as listed in Table 3. Since 4D weight tensor is reduced to matrix in real hardware implementation, column sparsity and row sparsity here represent filter-wise ($C \times K_\mathrm{h} \times K_\mathrm{w}$) and shape-wise($K_\mathrm{h} \times K_\mathrm{w}$) sparsity respectively. Layer sparsity is the overall sparsity which combines column and row in current layer. Different from SSL (Wen et al. 2016), we define the PE-wise($C_\mathrm{g} \times K_\mathrm{h} \times K_\mathrm{w}$) sparsity based on computation capacity of PE which is a straightforward concept that means the sparsity rate in a single group. As the simulation results listed in Table 3, we have relative 1.17% accuracy degradation and 16.2% sparsity reduction in comparison to SSL (Wen et al. 2016), but with a much higher compression rate.

ResNet we compare our work with existing non-structured weight quantization(i.e. binarization and ternarization) works. The inference accuracy results are listed in Table 4. Comparing with ABC-Net which uses multiple binarization approximation techniques, we could achieve almost the same accuracy with much higher compression rate. In addition, comparing with (He 2019) which uses the same non-structured ternarization method with our work, we achieve $1.35\times$ compression rate with even a little accuracy enhancement.

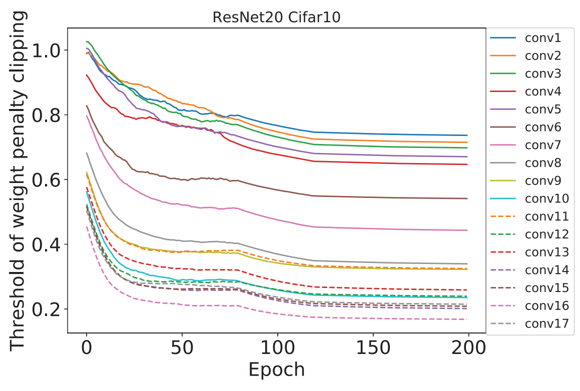
Ablation study and discussion

As proposed method is a combination of these two different compression techniques, to better evaluate the effectiveness, we do comparison with three different cases: weight ternarization, Group-Lasso pruning and directly combining these two techniques (naive combine) on both Cifar-10 and ImageNet dataset respectively. The results are shown in Table 5 and Table 6. Overall sparsity means the ratio of the individual zero values within the whole weight tensors. Differently, group sparsity represents the ratio of the number of PE-wise groups that are all zeros. We observe that when comparing with naive combine, our method achieves smaller overall sparsity, but larger useful group sparsity. This result demonstrates the effectiveness of the proposed method. By using weight penalty clipping with self-adapting threshold, only part of the weight groups which has smaller norm values will be regularized and rest of the weight groups keeps unchanged. focuses on the prune of unimportant weight groups and keeps rest of the weight groups unchanged.

Evolution of clipping threshold In this work, the value of clipping threshold and sparsity are two important factors which both are dynamic during the whole training processing. Fig. 5 shows the dynamics of sparsity and clipping threshold during training. It can be seen that the sparsity increases greatly at the first 80 epochs and then becomes coveraged. Meanwhile, the dynamic of clipping threshold has



(a)



(b)

Figure 5: Evolution of clipping threshold and sparasity

Table 6: Ablation study of ResNet-18 (He et al. 2016) on ImageNet.

	Acc	Overall sparsity	Group sparsity	Comp. rate
FP	69.75	-	-	1.0
Pruning	68.0	43	38	$\sim 1.6\times$
Tern	67.95	60	-	$\sim 16\times$
Naive combine	65.12	75	18	$\sim 19.5\times$
Ours	68.01	70	25	$\sim 21.3\times$

the similar pattern, the only difference is its value becomes smaller. It is because that the initial learning rate is large, which makes the Group Lasso regularizer have a strong effect on the weights. Moreover, we observe that later layers have larger sparsity than front layers.

PE capacity vs. structured sparsity ratio To evaluate the effectiveness of PE-group size, we select 5 different PE sizes on ResNet-20 (eg., PE-8 means the PE-size is $8 \times K_{\mathrm{h}} \times K_{\mathrm{w}}$), which are trained with the same hyper-parameter setting. As shown in Table 7, it can be seen that smaller PE-size is easier to be regularized which leads to larger sparsity and more accuracy loss under the same hyper-parameter setting.

Table 7: PE capacity versus structured sparsity ratio of ResNet-20 trained on CIFAR-10 dataset.

	PE-16	PE-8	PE-4	PE-2	PE-1
Accuracy (%)	90.89	90.56	90.29	89.93	89.66
Overall sparsity (%)	50	70.75	78.65	81.97	84.95
Group sparsity (%)	25	35.6	42.8	50.07	63.18

Relation to the norm based criterion (He et al. 2019) mentioned there are two requirements should be met for pruning: (1) the norm deviation of the filters should be large; (2) the minimum norm of the filters should be small. As shown in Fig.6, comparing with naive combine, norm distribution of our method has larger norm deviation, which means the norm distribution becomes suitable for pruning gradually during training. Obviously, it's easier to separate the important and unimportant weight groups according the absolute value of the norm for our method.

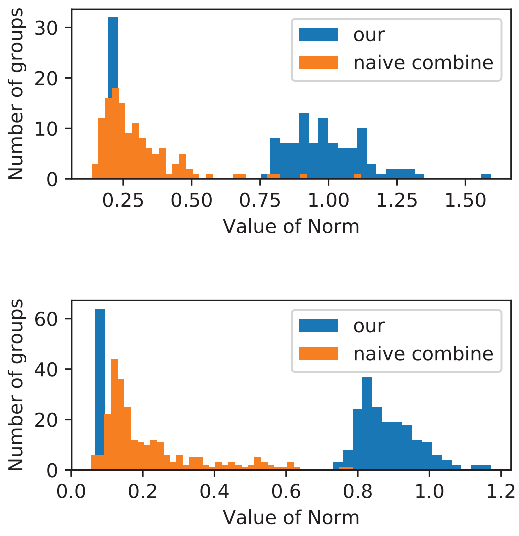


Figure 6: Norm based criterion on ResNet20 for CIFAR-10 dataset. (Top) is the conv8 layer and (Bottom) is conv15 layer.

FPGA implementation

To evaluate the performance of our proposed method in the real-word inference hardware accelerator, we deploy three representative convolutional layers of ResNet20 into a FPGA board. We compare our method with weight-ternarization-only method which also replaces MAC to add/sub operations, but without PE-wise pruning. To fairly compare, the bit-width of feature map is 16-bit fixed point number for both methods. The detailed setup and speedup is shown in Table 8. It is obvious that great speed-up is measured in real FPGA hardware implementation with our proposed PE-wise structured ternary network compared with non-structured ternary network.

Table 8: Convolutional layer implementation setup and speed up on FPGA

Layer	Weight size	PE size	Non-sparsity groups	Speedup
Stage 1	$(16, 16, 3, 3)$	$16 \times 3 \times 3$	9	1.56×
Stage 2	$(32, 32, 3, 3)$	$16 \times 3 \times 3$	33	1.69×
Stage 3	$(64, 64, 3, 3)$	$16 \times 3 \times 3$	119	1.73×

Conclusions

In this paper, we aim to integrate the Group Lasso based pruning and ternarization to maximum the efficiency of DNN hardware deployment. We firstly define the PE-wise sparsity, then based on the observation of the disharmony between these two methods, a new method is proposed, named weight penalty clipping with adjustable threshold to solve this problem. Moreover, comparing with related works in weight pruning and ternarization, we could achieve best state-of-the-art accuracy with a much higher compression rate.

Acknowledgments

This work is supported in part by the National Science Foundation under Grant No.1740126, No.1908495, No.1931871 and Semiconductor Research Corporation nCORE

References

Alvarez, J. M., and Salzmann, M. 2016. Learning the number of neurons in deep networks. In *Advances in Neural Information Processing Systems*, 2270–2278.

Bengio, Y.; Léonard, N.; and Courville, A. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*.

Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; and Fei-Fei, L. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, 248–255. Ieee.

Denton, E. L., et al. 2014. Exploiting linear structure within convolutional networks for efficient evaluation. In *NIPS*, 1269–1277.

Han, S.; Pool, J.; Tran, J.; and Dally, W. 2015. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, 1135–1143.

Han, S.; Mao, H.; and Dally, W. J. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*.

Hassibi, B., and Stork, D. G. 1993. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in neural information processing systems*, 164–171.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

He, Y.; Liu, P.; Wang, Z.; Hu, Z.; and Yang, Y. 2019. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 4340–4349.

He, Y.; Zhang, X.; and Sun, J. 2017. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, 1389–1397.

He, Z. e. a. 2019. Optimize deep convolutional neural network with ternarized weights and high accuracy. In *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, 913–921. IEEE.

Hinton, G., et al. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Hubara, I.; Courbariaux, M.; Soudry, D.; El-Yaniv, R.; and Bengio, Y. 2017. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research* 18(1):6869–6898.

Krizhevsky, A., and Hinton, G. 2009. Learning multiple layers of features from tiny images. Technical report, Citeseer.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 1097–1105.

Lebedev, V., and Lempitsky, V. 2016. Fast convnets using group-wise brain damage. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2554–2564.

LeCun, Y., et al. 2015. Deep learning. *nature* 521(7553):436.

LeCun, Y.; Denker, J. S.; and Solla, S. A. 1990. Optimal brain damage. In *Advances in neural information processing systems*, 598–605.

Leng, C.; Dou, Z.; Li, H.; Zhu, S.; and Jin, R. 2018. Extremely low bit neural network: Squeeze the last bit out with admm. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Li, H.; Kadav, A.; Durdanovic, I.; Samet, H.; and Graf, H. P. 2016. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*.

Lin, X.; Zhao, C.; and Pan, W. 2017. Towards accurate binary convolutional neural network. In *Advances in Neural Information Processing Systems*, 345–353.

Liu, B.; Wang, M.; Foroosh, H.; Tappen, M.; and Pensky, M. 2015. Sparse convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 806–814.

Liu, Z.; Li, J.; Shen, Z.; Huang, G.; Yan, S.; and Zhang, C. 2017. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE International Conference on Computer Vision*, 2736–2744.

Louizos, C.; Welling, M.; and Kingma, D. P. 2017. Learning sparse neural networks through l_0 regularization. *arXiv preprint arXiv:1712.01312*.

Molchanov, D.; Ashukha, A.; and Vetrov, D. 2017. Variational dropout sparsifies deep neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, 2498–2507. JMLR. org.

Rastegari, M., et al. 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, 525–542. Springer.

Srinivas, S., and Babu, R. V. 2015. Data-free parameter pruning for deep neural networks. *arXiv preprint arXiv:1507.06149*.

Sung, W.; Shin, S.; and Hwang, K. 2015. Resiliency of deep neural networks under quantization. *arXiv preprint arXiv:1511.06488*.

Wen, W.; Wu, C.; Wang, Y.; Chen, Y.; and Li, H. 2016. Learning structured sparsity in deep neural networks. In *Advances in neural information processing systems*, 2074–2082.

Yuan, M., and Lin, Y. 2006. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 68(1):49–67.