INTRAGROUP SPARSITY FOR EFFICIENT INFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

This work studies intragroup sparsity, a fine-grained structural constraint on network weight parameters. It eliminates the computational inefficiency of fine-grained sparsity due to irregular dataflow, while at the same time achieving high inference accuracy. We present theoretical analysis on how weight group sizes affect sparsification error, and on how the performance of pruned networks changes with sparsity level. Further, we analyze inference-time I/O cost of two different strategies for achieving intragroup sparsity and how the choice of strategies affect I/O cost under mild assumptions on accelerator architecture. Moreover, we present a novel training algorithm that yield models of improved accuracies over the standard training approach under the intragroup sparsity constraint.

1 INTRODUCTION

State-of-the-art performance of deep neural networks in many application domains has set off the trend in real-world deployment of these models. Models of superior performance are often of high computing complexity due to large network widths and depths. The resulting high cost and high latency are often prohibitive for resource-limited devices.

Various methods have been proposed to produce lightweight networks to improve computational efficiency while maintaining satisfactory model performance (Han et al., 2015; Howard et al., 2017). A widely used technique, *network pruning* removes unimportant weights to yield sparse models of potentially lower computational complexity. Pruning can be either *fine-grained*, *i.e.* individual weights are independently subject to removal (Han et al., 2015), or *structured* (*coarse-grained*), *i.e.* weights are removed in groups, such as entire channels or blocks of weights inside a convolutional filter (Mao et al., 2017). Fine-grained pruning typically yields models with higher parameter efficiency, but often does not improve computational efficiency at inference time due to the irregularity in data flow resulting from haphazard sparsity patterns. Structured pruning, however, can improve computational efficiency, but seldom reach comparable levels of parameter efficiency (Mao et al., 2017).

To push beyond the frontier set by this tradeoff and achieve both high accuracy and high computational efficiency, this work studies *intragroup sparsity* (Fig. 1a). In contrast to unconstrained fine-grained sparse network, intragroup-sparse weight tensors are subdivided into small groups such that active weight pruning yields a fixed number of nonzero weights in each group. The groups are arranged to be contiguous along the dimension of either output or input channels. As such, the intragroup-sparse structural constraint eliminates the data inflow irregularity associated with unconstrained fine-grained sparse networks, which, with proper hardware acceleration as recently revealed by Pool (2020), can realize speedup of fine-grained sparse network inference that scales linearly with the model sparsity level. Using a novel pruning-during-training algorithm, we show that the weight group size can be made small (such as 8 and 16) without significantly degrading the accuracy of a model.

Contributions:

- We present theoretical analysis on how weight group size and sparsity level affect intragroup fine-grained weight sparsification error and the resulting generalization performance of the model.
- We propose a training solution that overcomes the difficulty of optimizing intragroup sparse networks (Yao et al., 2019) with small weight group sizes. Our method produces models that outperform state-of-the-art lightweight architectures and pruned models with structured sparsity.

- We analyze inference I/O cost of cross-channel (CC) sparsity and bank-balanced (BB) sparsity (Yao et al., 2019), two different approaches for achieving intragroup sparsity, and demonstrate the advantage of CC sparsity for hardware acceleration under mild assumptions.

The rest of this article is organized as follows. Sec. 2 summarizes relevant literature. Sec. 3 defines the intragroup sparsity structure and explains the resulting efficiency for inference. Sec. 4 gives a theoretical analysis on how the intragroup sparsity constraint affects model performance. Sec. 5 describes the pruning algorithm designed to optimize performance under the intragroup sparse constraint, with a demonstration of the advantage of our pruning algorithm over the iterative pruning method employed by Yao et al. (2019). Sec. 6 summarizes empirical experimental results of training networks of intragroup sparsity versus other structured sparse and lightweight architectures. Sec. 7 concludes the article.

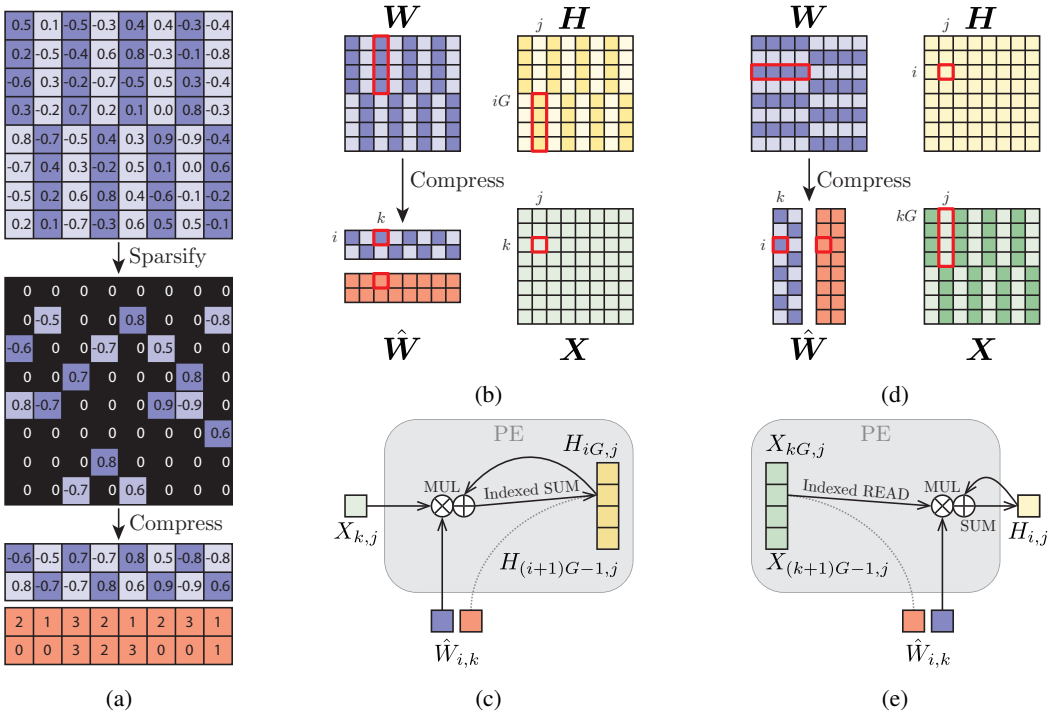


Figure 1: The CC sparsity constraint and the associated accelerator datapath for general matrix multiplication (GEMM), in comparison with BB sparsity. (a) An illustration of CC sparsity imposed on an 8×8 weight matrix with $G = 4, s = 1$. See main text for details. (b) GEMM with a CC-sparse weight matrix, illustrated for the same pattern as in (a). (c) CC-sparse GEMM dataflow for the case in (b). Illustrated is a single operation involving elements highlighted with red rectangles in (b). Partial sum $\boldsymbol{H}$ is persistent in PE's register (output-stationary). (d, e) Similar to (b, c), for BB sparsity. See main text for details.

2 Related work

Lightweight convolutional neural network (CNN) architectures: Recent lightweight architectures such as MobileNet (Howard et al., 2017) and MobileNetV2 (Sandler et al., 2018), decompose convolutional layers into filters along different spatial dimensions. Specifically, they employ depth-wise separable convolutions, reducing the computation complexity. Group convolution (Gao et al., 2018; Huang et al., 2018; Ma et al., 2018; Zhang et al., 2017) goes one step further by separating convolution operations into groups, removing the connections across filter groups. The proposed intragroup sparsity structure has the advantage over group convolution in that it reduces input dimensionality without suffering from the strong constraint of separating input into groups.

Sparse CNNs: Reducing network redundancy through pruning (or sparsification) of a large network is an alternative of creating compact networks. Unstructured pruning removes individual network weights that meet certain criteria (Han et al., 2015) and generates networks of fine-grained sparsity, a technique shown effective in considerable reduction of model sizes. But fine-grained sparse models are often hardware-inefficient at inference time. Structured pruning removes network connections under constraints that allow efficient model inference (Li et al., 2016; Mao et al., 2017; Narang et al., 2017; Wen et al., 2016). However, structured pruning procedures do not offer the same level of parameter efficiency as unstructured pruning (Mao et al., 2017; Yao et al., 2019).

Intragroup sparsity: Intragroup sparsity (Zhou et al., 2010) is originally proposed as a form of model regularization, where the $\ell_{1,2}$-norm is used to induce sparsity within a group of model parameters for feature selection. Wu et al. (2018) adopts intragroup sparsity structure to overcome the irregularity associated with their dendritic neural networks; their weight group is constructed across dendritic subkernels with *fixed* connection maps. In this study, the cross-channel (CC) weight groups are constructed across network channels with *learned* connection maps.

Bank-balanced sparsity: A different approach to the same end is bank-balanced (BB) sparsity (Yao et al., 2019), also employed by recently unveiled Nvidia Ampere GPU architecture (Pool, 2020). In contrast to CC sparsity, BB-sparse weight groups are contiguous blocks from different input channels instead of output channels (*i.e.*, weights from the same rows instead of columns in Fig. 1d). Compared to ours, the group sizes chosen in Yao et al. (2019) are much larger, a much weaker constraint less suited for high accelerator speedup. Further, here we describe a training algorithm that significantly outperforms the iterative pruning technique employed by Yao et al. (2019), yielding more efficient models with small group sizes (see Sec. 3.3). Moreover, here we show that CC sparsity, under mild architectural assumptions, is more conducive for specialized hardware acceleration.

3 INTRAGROUP SPARSITY

3.1 THE INTRAGROUP SPARSITY STRUCTURE

Pruned networks with unstructured, fine-grained sparsity patterns often lead to model inference inefficiency on popular hardware accelerators (Cao et al., 2019; Han et al., 2016; Mao et al., 2017; Parashar et al., 2017; Zhang et al., 2016; Zhou et al., 2018). To address the computing inefficiency associated with fine-grained sparse networks, we study the intragroup sparse structural constraint, a sparsity pattern applicable to both convolutional and dense layer parameters.

For clarity, let us use a fully-connected network layer with CC sparsity as an example to explain the concept. We represent its input by column vector $\boldsymbol{x} \in \mathbb{R}^N$, its weights by a matrix $\boldsymbol{W} \in \mathbb{R}^{M \times N}$ and its output column vector $\boldsymbol{h} \in \mathbb{R}^M$, *s.t.* $\boldsymbol{h} = \boldsymbol{W} \times \boldsymbol{x}$ is satisfied. For example, for the weight matrix $\boldsymbol{W}$ illustrated in Fig. 1a, we have $M = 8, N = 8$, and each row of $\boldsymbol{W}$ contains weights for one output channel in $\boldsymbol{h}$. To impose intragroup sparsity constraint on $\boldsymbol{W}$, it is first partitioned into groups. For each weight group, a fixed number of weights are set to zero by sparsification; see Fig. 1a for an example of pruning with a weight group size of $G = 4$ and $s = 1$ nonzero weights per group. This means the weights for each group are from 4 channels (rows in $\boldsymbol{W}$) among which there is 1 nonzero component. As such, we can compress every $G = 4$ rows in sparsified $\boldsymbol{W}$ into exactly $s = 1$ rows of compressed weights, yielding $\hat{\boldsymbol{W}}$. Each new row in $\hat{\boldsymbol{W}}$ has the same number of columns as the original weight matrix. To identify the original row within which a weight in $\hat{\boldsymbol{W}}$ is located in $\boldsymbol{W}$, a row index accompanies each weight in the compressed matrix (this index specifies to which row in $\boldsymbol{h}$ we add the multiplication result). For this example of $G = 4$, we require a 2-bit index for each weight (shown as the red matrix at the bottom of Fig. 1a). In general, one needs $s \log_2 G$ index bits for each weight group–even though the theoretical number of index bits is slightly smaller, it is practical to use the plain index to avoid extra decoding overhead. Small group sizes incur lower overhead in the extra storage and the I/O requirements associated with weight indices, while at the same time place stronger constraints on the model, yielding lower accuracy (see Sec. 5).

3.2 EFFICIENT INFERENCE WITH INTRAGROUP SPARSE NETWORKS

To demonstrate that intragroup sparsity can eliminate irregular data access so as to enable efficient neural network inference, let us consider the standard general matrix-matrix multiplication (GEMM),

the core computation routine heavily used in neural network inference for both convolutional and fully-connected network architectures (Chetlur et al., 2014; Dukhan, 2019). Specifically, compute $\boldsymbol{H} = \boldsymbol{W} \times \boldsymbol{X}$ with weight matrix $\boldsymbol{W}$, input $\boldsymbol{X}$, and multiplication result $\boldsymbol{H}$. First, we consider $\boldsymbol{W}, \boldsymbol{X}, \boldsymbol{H} \in \mathbb{R}^{N \times N}$ in their dense forms, as shown in Fig. 1b. We present a naive form of the GEMM as in Alg. 1. The number of computational steps of the naive procedure for this case is N^3.

Algorithm 1: Dense matrix-matrix multiplication. Access to $\boldsymbol{H}, \boldsymbol{W}$, and $\boldsymbol{X}$ is sequential, with $\boldsymbol{H}, \boldsymbol{W}$ stored in row major, $\boldsymbol{X}$ in column major.

Require: $\boldsymbol{W}, \boldsymbol{X} \in \mathbb{R}^{N \times N}$, $\boldsymbol{H} = 0$.
Ensure: $\boldsymbol{H} = \boldsymbol{W} \times \boldsymbol{X}$
  for $i \leftarrow 0$ to $N - 1$ by 1
    for $j \leftarrow 0$ to $N - 1$ by 1
      for $k \leftarrow 0$ to $N - 1$ by 1
        $H_{i,j} \leftarrow H_{i,j} + W_{i,k} X_{k,j}$

Algorithm 2: Intragroup-sparse matrix-matrix multiplication. Access to $\hat{\boldsymbol{W}}, \boldsymbol{X}$ is sequential, with $\boldsymbol{H}, \hat{\boldsymbol{W}}$ stored in row major, $\boldsymbol{X}$ in column major.

Require: $\hat{\boldsymbol{W}}, \boldsymbol{X} \in \mathbb{R}^{N \times N}$, $\boldsymbol{H} = 0$.
Ensure: $\boldsymbol{H} = \hat{\boldsymbol{W}} \times \boldsymbol{X}$
  for $i \leftarrow 0$ to $N/G - 1$ by 1
    for $j \leftarrow 0$ to $N - 1$ by 1
      for $k \leftarrow 0$ to $N - 1$ by 1
        $(I, V) \leftarrow (\hat{W}_{i,k}[\text{index}], \hat{W}_{i,k}[\text{value}])$
        $H_{Gi+I,j} \leftarrow H_{Gi+I,j} + V X_{k,j}$

Next, let us compress the weight matrix $\boldsymbol{W}$ into a CC intragroup-sparse format $\hat{\boldsymbol{W}}$. In this case, if we assume a group size of $G = 4$ with $s = 1$, we reduce the size of the weight matrix by a factor of $G/s = 4$ times. Each element in $\hat{\boldsymbol{W}}$ now has two components: a weight value and an index. We denote them by $\hat{W}_{i,k}[\text{value}]$ and $\hat{W}_{i,k}[\text{index}]$, respectively. The GEMM computation is then Alg. 2 (for clarity, we assume $s = 1$ here). With intragroup sparsity, we reduce the number of computational steps by a factor of G to N^3/G. At the same time, the dataflow of reading $\hat{\boldsymbol{W}}$ and $\boldsymbol{X}$ remains continuous and regular, as in the dense case. The dataflow of matrix $\boldsymbol{H}$ is also continuous at the group level. Irregularity only remains inside each step when the index I associated with each weight is used to access the corresponding element inside the output group. This can be overcome through making a whole group of elements of $\boldsymbol{H}$ persistent in on-chip registers, thereby circumventing the latency and inefficiency caused by irregular off-chip memory access Cao et al. (2019).

Thus, intragroup sparsity constraint improves the dataflow of unstructured fine-grained sparse networks. Furthermore, thanks to the introduction of the weight group structure, intragroup sparsity enables splitting execution along the boundaries of weight groups, enabling parallel processing and matrix-tilling-based data reuse for further improvement of inference efficiency.

3.3 CROSS-CHANNEL SPARSITY *vs.* BANK-BALANCED SPARSITY

The intragroup-sparse structure we introduced above build weight groups across different network output channel, we refer to it as cross-channel (CC) sparsity. Similarly, in Cao et al. (2019); Yao et al. (2019), the authors propose and demonstrate the performance of the complementary, bank-balanced (BB) sparsity structure. CC sparsity is similar to BB sparsity in that both introduce uniformly sized weight groups in network parameters, and both associate an index with each compressed weight. The central difference between the two structures is illustrated in Fig. 1: CC-sparse weight groups span different output channels ($G = 4$ contiguous weights of the same color, as in Fig. 1b. For example, $W_{0,0}, W_{1,0}, W_{2,0}$, and $W_{3,0}$ form a weight group, and they are compressed into a single weight in $\hat{\boldsymbol{W}}$ as $\hat{W}_{0,0}$. Assume $W_{2,0}$ is the weight that survives the pruning process; that is, a weight index of 2 is associated with $\hat{W}_{0,0}$. Then, the weight value of $\hat{W}_{0,0}$ is multiplied with the $X_{0,j}$, and accumulated to the corresponding $H_{2,j}$. This computational procedure is illustrated in Fig. 1c.

In contrast, for BB sparsity (Yao et al., 2019), weight groups are formed by weights that correspond to different inputs ($G = 4$ contiguous weights of the same color, as illustrated in Fig. 1d). Thus, the index associated with the compressed weights is used to allocate the corresponding input elements inside a group, as illustrated in Fig. 1e.

A number of recent studies on specialized neural network accelerators show that data I/O, namely reading/writing data from/to off-chip memory, dominates the total energy budget (Zhou et al., 2018). Dataflow designs differ drastically among existing accelerators so as to optimize system efficiency for specific use cases (Deng et al., 2020). Though a thorough I/O complexity analysis (Jia-Wei & Kung, 1981) for either CC or BB sparsity on all accelerator designs is beyond the scope of this study,

we analyze an important case with mild assumptions on accelerator architecture to demonstrate a clear advantage of CC over BB sparsity.

In this analysis, we assume that each processing element (PE) contains an adequate number of registers necessary for the implementation of GEMM involving either a CC- or a BB-sparse weight matrix. No matrix tiling or other data parallelism is considered here. For the CC-sparse case, we use the output stationary dataflow; that is, for each stride, we set up G registers for the output accumulation results of a group (contiguous blocks of the same color, as in matrix $\boldsymbol{H}$ of Fig. 1c). We keep those G registers persistent for maximum reuse. Inside a stride, for each calculation step, we read one element from $\hat{\boldsymbol{W}}$ and one element from $\boldsymbol{X}$. There are N steps inside each stride. At the end of a stride, we write G elements of $\boldsymbol{H}$ into host memory. The number of groups inside $\boldsymbol{H}$ is N^2/G. Thus, the total memory I/O count for the multiplication is $(2N + G) \times N^2/G = 2N^3/G + N^2$.

In the case of BB sparsity, for each stride (Fig. 1e), we first read G elements from $\boldsymbol{X}$ and store them in PE registers for maximum reuse. Then, for each computational step, we read one element from $\hat{\boldsymbol{W}}$, and one from $\boldsymbol{H}$. We not only need to read one element but also to write it, because it contains a partial summation result, only except for the first step when every element in $\boldsymbol{H}$ is zero. Finally, we have the same number of strides to that in the CC case. Therefore, for the BB-sparse case, the memory I/O count is $(3N + G) \times N^2/G - N^2 = 3N^3/G$. The difference of I/O requirements between these two cases is $N^3/G - N^2$. Since N is typically much larger than G, CC sparsity can significantly reduce the I/O complexity in this case.

4 Impact of intragroup sparsity constraints

Because intragroup sparsity imposes a structural constraint on the network model, it limits model capacity. In this section we present both theoretical and empirical analyses to assess this impact.

Consider a pre-trained network layer subject to the intragroup-sparse constraint. Here we use a fully connected network layer for the sake of clarity. We denote by $\boldsymbol{W}$ the weight matrix in dense form, where each row of $\boldsymbol{W}$ corresponds to one output channel. Assume that each weight element in $\boldsymbol{W} \in \mathbb{R}^{M \times N}$ independently has the same probability P_s (sparsity level) of being zero. Partition the N weights of each row into $g = \frac{N}{G}$ groups, each group of size G. Denote by $s = G(1 - P_s)$ the number of slots available for the storage of the nonzero weight values in each group. For simplicity, assume $g, G, s \in \mathbb{N}^+$. Given each weight in $\boldsymbol{W}$ has the same probability of being nonzero, the number of nonzero weights i falling in each weight group is

$$\binom{G}{i}(1 - P_s)^i P_s^{G-i}. \tag{1}$$

If $i > s$ for a weight group, then $i - s$ nonzero weights must be discarded. Thus,

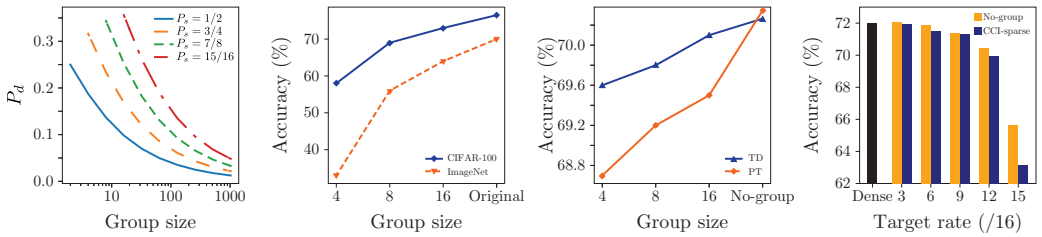
$$P_d = \frac{\sum_{i=1}^{G} \mathbb{1}(i > s)(i - s)\mathcal{B}(i, G, 1 - P_s)}{s} \tag{2}$$

is the probability of any weight failing to be assigned to an encoding slot; here $\mathbb{1}(\cdot)$ is the indicator function. By Eqn. 2, we plot the relationship among P_d, the group size G and the sparsity ratio P_s in Fig. 2a. **For the same sparsity P_s, P_d steadily increases as the group size G decreases. In addition, for the same group size G, P_d is significantly larger at a higher sparsity level P_s.**

To gain insight into how the group size affects model accuracy, we apply the intragroup sparsity structure on a pre-trained MobileNetV2 with a sparsity ratio of 75% (sparse pointwise layers only, as most of compute is in the pointwise operations). As illustrated in Fig. 2b, while we aim at enforcing the same sparsity ratio of 75% on models, a smaller group size causes more severe performance degradation, correlated with the rising allocation error rate when we decrease the group size.

5 Training intragroup-sparse models

The abovementioned results (Fig. 2b) are obtained by post-training imposition of intragroup sparsity on pre-trained models, models optimized without the intragroup sparsity constraint during training. In this section, we present an algorithm that trains intragroup-sparse networks from scratch.

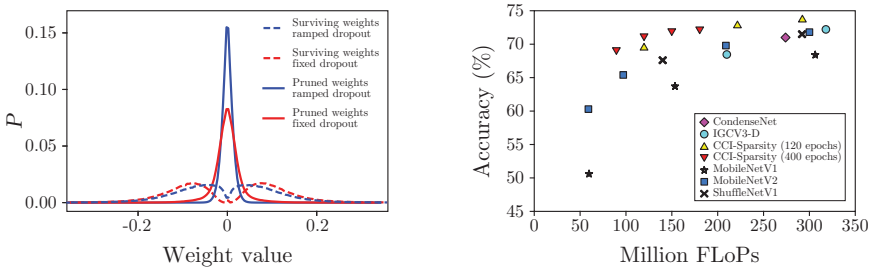


(a) (b) (c) (d)

Figure 2: Impact of the intragroup sparsity constraint. (a) Probabilities of a nonzero weight failing to be allocated an encoding slot, for various group sizes and sparsity levels P_s of $\frac{1}{2}$, $\frac{3}{4}$, $\frac{7}{8}$, and $\frac{15}{16}$. The group sizes G plotted are $2, 4, \cdots, 1024$. (b) The effect of group size ($G = 4, 8,$ and 16, $s = 1, 2,$ and 4, respectively) on accuracy (for CIFAR-100 and ImageNet) of a pre-trained MobileNetV2 intragroup-sparsified to 75% sparsity ratio. *Original*: model not subject to the intragroup sparsity structure. We re-calibrate the batch-normalization statistics after model pruning, but no model fine-tuning is conducted. (c) A comparison between TD and PT pruning in generating intragroup-sparse networks. *No-group*: no intragroup sparsity imposed. (d) The effect of sparsity level on model accuracy and a comparison between models with and without intragroup sparsity constraint. Group size $G = 16$. *Dense*: the original dense model.

There are two classes of approaches to training a sparse neural network. One is to prune a pre-trained dense network, followed by post-training (PT) fine-tuning (Han et al., 2015; Zhu & Gupta, 2017; Yao et al., 2019); this approach is used by Yao et al. (2019) for training BB-sparse networks. The other is to learn the sparse structure directly; *e.g.* targeted dropout (TD) (Gomez et al., 2018) and dynamic sparse reparameterization (Mocanu et al., 2018; Mostafa & Wang, 2019) belong to this category. Both approaches are known to yield sparse neural network models of similar performance.

As analyzed above, the strength of the intragroup sparsity constraint is negatively correlated with the group size G. A small group size (ideal for hardware acceleration), *e.g.* $G = 8$, imposes a strong constraint on model capacity (see Fig. 2). Since the probability of any important weights being pruned away is high with small group sizes, we hypothesize that models would have higher accuracy if they were given opportunities to recover pruned weights and to fine-tune to the intragroup-sparse constraint. In contrast, during iterative PT pruning, a pruned weight does not have the chance of being recovered. Our hypothesis predicts that PT pruning would yield models of inferior performance than TD. We test the hypothesis with experiments described in the following.



(a) (b)

Figure 3: (a) The weight distributions in the original (*red*) and proposed (*blue*) dropout ramping model training. *Solid lines*: weights from the pruning pool. *Dashed lines*: weights protected from pruning. The dropout ratio ramping causes more weights in the pruning pool to shrink toward 0. (b) Efficiency-accuracy tradeoff for the ImageNet classification task compared among various lightweight and pruned network architectures. The horizontal axis corresponds to the total operation counts (in FLoPs). For MobileNetV1, MobileNetV2, and ShuffleNetV1, we also include the results from models with the width multiplied by various scales. MobileNetV2 with intragroup sparsity: *Yellow triangles:* $G = 8$, $s = 2$, with width scales of 1.0, 1.4 and 2.0, trained for 120 epochs; *Red inverted triangles:* $G = 8$ with $s = 1, 2, 3$ and 4, trained for 400 epochs.

5.1 IMPROVED TARGETED DROPOUT

We propose an improved version of TD for training intragroup-sparse models. TD training begins with a dense network structure. During the course of training, a set of candidate network connections are selected based on a specific policy (in our case, we target weights of low absolute magnitude for pruning). Then, candidate connections are dropped with a specified probability similar to dropout (Srivastava et al., 2014).

In the original TD paper (Gomez et al., 2018), the authors increase the size of the dropout candidate pool during training and use a fixed 50% dropout ratio. While their approach is successful on small datasets reported, we find that this strategy does not yield models of competitive accuracy when applied to training on the ImageNet dataset Russakovsky et al. (2015). Gomez et al. (2018) suggest that TD causes the unimportant connections in the pruning pool to shrink toward zero due to ℓ_2 regularization. This property is important, as the weights in the pruning pool participate in the model training with a fixed probability, and if those weights are of significant magnitude, they would affect the running mean and variance statistics of the batch-normalization layers in the network, leading to inferior model performance (Li et al., 2018).

As shown in Fig. 3a, the pruned weights are actually of significant magnitudes in our model. Ideally, those pruned weights should stay at the value of exact zero such that they would not affect batch-normalization. A possible mitigation is to use a higher dropout rate for the TD, giving weights in the pruning pool a higher chance to shrink toward zero. However, this also lowers the chance of recovery of a pruning candidate weight. To solve this problem, we schedule the candidate dropout rate from an initial 50% toward 100% to make the candidate weights eventually shrink towards 0 while allowing the weights a higher chance of recovery at the early stage of model training. As shown in Fig. 3a, the dropout rate ramping indeed causes more pruned weights to shrink toward zero. This improvement leads to significantly higher model test accuracy. In a typical case with $G = 8$ and $s = 2$, a 75% sparsity ratio, a MobileNetV2 model trained on the ImageNet dataset reaches a top-1 accuracy of 69.8% (ramping dropout) *vs.* 64.8% (fixed 50% dropout rate).

Appropriate model regularization can often lead to improved generalization performance (Li et al., 2016; Wen et al., 2016). For the main part of this study, we use a simple ℓ_2 regularizer and rely on TD to induce intragroup sparsity. Our preliminary investigation (Appendix A.2) suggests that exclusive Lasso regularization can improve model training and yield higher test accuracy. Further investigation is necessary to understand the interaction between exclusive Lasso regularization and TD training.

5.2 CHOICES OF GROUP SIZE FOR TARGETED DROPOUT *vs.* ITERATIVE PRUNING

For intragroup sparsity, small group sizes require less weight index storage, as well as smaller register file sizes, improving data locality that leads to smaller area and less wiring in hardware design and a lower energy cost for register file accessing Yang et al. (2018). However, as explained in Sec. 3, small group sizes impose a strong constraint on models and lead to inferior performances–a tradeoff. How do TD and PT training compare with each other in terms of tolerating small group sizes?

To address this question, we experiment with MobileNetV2 models with all pointwise layer sparsity ratio at 75% for various group sizes, trained with TD and with PT. As longer training typically leads to higher model accuracy, we set the number of training epochs such that model in the *no-group* setting has very similar accuracies under TD and PT, to make a fair comparison. For TD, all models are trained for 120 epochs. For PT, the models are first trained for 120 epochs and then iteratively pruned for additional 60 epochs. As shown in Fig. 2c, smaller weight group sizes indeed cause more accuracy loss than a large group size (similar behavior is observed with BB sparsity, results not shown). More importantly, though PT yields similar accuracy to TD when no intragroup sparsity is imposed (*no-group*), accuracy of models trained with TD degrade much more gracefully with increasing sparsity than those trained with PT, even though PT involves additional epochs of iterative pruning.

5.3 EFFECT OF SPARSITY LEVELS

As revealed in Fig. 2a, the probability of a model to allocate an encoding space successfully for a certain nonzero weight decreases with increasing sparsity level. Here we verify this theoretical conjecture empirically. As shown in Fig. 2d, we observe diminished accuracies when models are

pruned to higher sparsity levels. Models trained with *vs.* without the intragroup sparsity constraint at medium and low sparsity levels did not differ significantly in accuracy. Perhaps, the structural constraint also biases models toward better generalization performance under mild sparsity level as suggested in Zhou et al. (2010). A substantial accuracy drop happens at sparsity level of $\frac{15}{16}$, consistent with our theoretical prediction.

6 EXPERIMENTAL RESULTS

In this section, we employ a combination of the abovementioned techniques, in attempt to produce intragroup-sparse networks with as low computing complexity as possible at given accuracies. Experiments are conducted on the ImageNet Russakovsky et al. (2015) and CIFAR-10 datasets Krizhevsky & Hinton (2009). To demonstrate the applicability of intragroup sparsity to various network architectures, we prune both heavyweight models, such as VGG and ResNet, and also a more parameter efficient model, MobileNetV2. For details of the experimental setting see Appendix A.1.

Results on the CIFAR-10 dataset: The results of ResNet-32 and VGG-16 models trained on the CIFAR-10 dataset are presented in Tabs. A.1 ,A.2 (see Appendix). We further compare our result of VGG-16 with a structured sparse model as described in Li et al. (2016), where entire filters are targeted for pruning. intragroup-sparse models also show a clear advantage in this case.

Results on the ImageNet dataset: For ImageNet, we first compare the test accuracy of intragroup-sparse MobileNetV2 (pruned pointwise layer only) with that of lightweight architectures from the latest literature (Huang et al., 2018; Sun et al., 2018; Ma et al., 2018; Howard et al., 2017) in Fig. 3b. The results of different model sizes from these reports are shown here. For a fair comparison, we scale up the model size (yellow triangles), and train for 120 epochs as in standard setting. We also show results (red inverted triangles) from models of different sparsity levels (trained for 400 epochs for better performance). Evidently, intragroup-sparse models trained with improved TD (ours) outperform those lightweight architectures by a significant margin (see Tab. A.4 in Appendix for details). Finally, we compare intragroup sparsity with coarse-grained sparsity using ResNet-18 (Dong et al., 2017; He et al., 2018) trained on ImageNet, as shown in Tab. A.3 in Appendix. Again, intragroup-sparse models considerably outperform structured pruning.

7 CONCLUSIONS

This work analyzes the intragroup sparsity structure, a hardware accelerator-friendly parameter constraint, as well as an effective algorithm to train models under such constraints. Our method retains the performance advantage of fine-grained sparsity over coarse-grained structured pruning, while at same time, avoids the inference inefficiency of fine-grained sparsity due to irregularity in the computing dataflow. Additionally, intragroup sparsity is compatible with matrix-tiling often used in accelerator designs, enabling higher inference efficiency than fine-grained sparsity through data reuse and parallelism.

Through theoretical and empirical analyses, we demonstrate the tradeoff between computational efficiency and the granularity of the structural constraint, controlled by the group size G. While a smaller group size incurs lower computing overhead, it has a stronger impact on model performance. Our findings suggest that a weight group size of 16 does not typically cause significant performance loss. Our theoretical analysis shows that models with a higher sparsity level are more strongly affected by the group structure. Experimental results show that it is better to train compact models with a proper sparsity level rather than very wide networks with very high sparsity.

Trained with the same algorithm and hyperparameters, both CC- and BB-sparse models achieve comparable accuracies. However, we present a commonplace scenario in hardware acceleration where CC sparsity requires lower I/O access than BB sparsity for model inference.

Our proposed model training method outperforms iterative pruning (as used by Yao et al. (2019)) in training intragroup-sparse models. Finally, compared with several lightweight network architectures and structural pruning, sparse networks produced by our method outperforms the state-of-the-art in terms of best inference accuracy under a given computational budget.

REFERENCES

Shijie Cao, Chen Zhang, Zhuliang Yao, Wencong Xiao, Lanshun Nie, Dechen Zhan, Yunxin Liu, Ming Wu, and Lintao Zhang. Efficient and effective sparse lstm on fpga with bank-balanced sparsity. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 63–72. ACM, 2019.

Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*, 2014.

Lei Deng, Guoqi Li, Song Han, Luping Shi, and Yuan Xie. Model compression and hardware acceleration for neural networks: A comprehensive survey. *Proceedings of the IEEE*, 108(4):485–532, 2020.

Xuanyi Dong, Junshi Huang, Yi Yang, and Shuicheng Yan. More is less: A more complicated network with less inference complexity. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

Marat Dukhan. The indirect convolution algorithm. *arXiv preprint arXiv:1907.02129*, 2019.

Hongyang Gao, Zhengyang Wang, and Shuiwang Ji. Channelnets: Compact and efficient convolutional neural networks via channel-wise convolutions. In *Advances in Neural Information Processing Systems*, pp. 5203–5211, 2018.

Aidan N Gomez, Ivan Zhang, Kevin Swersky, Yarin Gal, and Geoffrey E Hinton. Targeted dropout. *NIPS 2018 CDNNRIA Workshop*, 2018.

Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pp. 1135–1143, 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. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pp. 243–254. IEEE, 2016.

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, 2016.

Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. Soft filter pruning for accelerating deep convolutional neural networks. *CoRR*, abs/1808.06866, 2018. URL http://arxiv.org/abs/1808.06866.

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

Gao Huang, Shichen Liu, Laurens Van der Maaten, and Kilian Q Weinberger. Condensenet: An efficient densenet using learned group convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2752–2761, 2018.

Hong Jia-Wei and H. T. Kung. I/o complexity: The red-blue pebble game. In *Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing*, STOC '81, pp. 326–333, New York, NY, USA, 1981. Association for Computing Machinery. ISBN 9781450373920. doi: 10.1145/800076.802486. URL https://doi.org/10.1145/800076.802486.

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

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

Xiang Li, Shuo Chen, Xiaolin Hu, and Jian Yang. Understanding the disharmony between dropout and batch normalization by variance shift. *arXiv preprint arXiv:1801.05134*, 2018.

Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 116–131, 2018.

Huizi Mao, Song Han, Jeff Pool, Wenshuo Li, Xingyu Liu, Yu Wang, and William J Dally. Exploring the regularity of sparse structure in convolutional neural networks. *arXiv preprint arXiv:1705.08922*, 2017.

Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H Nguyen, Madeleine Gibescu, and Antonio Liotta. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature communications*, 9(1):2383, 2018.

Hesham Mostafa and Xin Wang. Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. *arXiv preprint arXiv:1902.05967*, 2019.

Sharan Narang, Eric Undersander, and Gregory Diamos. Block-sparse recurrent neural networks. *arXiv preprint arXiv:1711.02782*, 2017.

Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W Keckler, and William J Dally. Scnn: An accelerator for compressed-sparse convolutional neural networks. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pp. 27–40. IEEE, 2017.

Jeff Pool. Accelerating sparsity in the nvidia ampere architecture. *GTC 2020*, 2020.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4510–4520, 2018.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1): 1929–1958, 2014.

Ke Sun, Mingjie Li, Dong Liu, and Jingdong Wang. Igcv3: Interleaved low-rank group convolutions for efficient deep neural networks. *arXiv preprint arXiv:1806.00178*, 2018.

Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Advances in neural information processing systems*, pp. 2074–2082, 2016.

Xundong Wu, Xiangwen Liu, Wei Li, and Qing Wu. Improved expressivity through dendritic neural networks. In *Advances in neural information processing systems*, pp. 8057–8068, 2018.

Xuan Yang, Mingyu Gao, Jing Pu, Ankita Nayak, Qiaoyi Liu, Steven Emberton Bell, Jeff Ou Setter, Kaidi Cao, Heonjae Ha, Christos Kozyrakis, et al. Dnn dataflow choice is overrated. *arXiv preprint arXiv:1809.04070*, 2018.

Zhuliang Yao, Shijie Cao, Wencong Xiao, Chen Zhang, and Lanshun Nie. Balanced sparsity for efficient dnn inference on gpu. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 5676–5683, 2019.

Shijin Zhang, Zidong Du, Lei Zhang, Huiying Lan, Shaoli Liu, Ling Li, Qi Guo, Tianshi Chen, and Yunji Chen. Cambricon-x: An accelerator for sparse neural networks. In *The 49th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 20. IEEE Press, 2016.

Ting Zhang, Guo-Jun Qi, Bin Xiao, and Jingdong Wang. Interleaved group convolutions. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 4373–4382, 2017.

Xuda Zhou, Zidong Du, Qi Guo, Shaoli Liu, Chengsi Liu, Chao Wang, Xuehai Zhou, Ling Li, Tianshi Chen, and Yunji Chen. Cambricon-s: Addressing irregularity in sparse neural networks through a cooperative software/hardware approach. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 15–28. IEEE, 2018.

Yang Zhou, Rong Jin, and Steven Chu-Hong Hoi. Exclusive lasso for multi-task feature selection. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pp. 988–995, 2010.

Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*, 2017.

A APPENDIX

A.1 EMPIRICAL EXPERIMENT RESULT

For ImageNet experiments, we use the ILSVRC-2012 subset. The models are trained with the augmented standard training set and evaluated with the center crop of images from the validation set, as described in MobileNetV2 Sandler et al. (2018). Unless explicitly specified, the models are trained with the SGD optimizer with momentum of 0.9 and an initial learning rate of 0.18, on 4 GPUs with a batch size of 64 per GPU for a total of 120 epochs with the cosine decay schedule Loshchilov & Hutter (2016). For standard models, the weight decay is set to 0.00004. For fair comparison, we do not tune any of the hyperparameters that are described above throughout our experiments. For TD training, we ramp up the targeted rate while keeping the candidate dropout rate at 50% during the first half of training epochs, and then ramp up the candidate dropout rate from 50% to 100% during the second half. We compress convolutional layers in a network with the same group size G and s setting. For MobileNetV2, we only compress the pointwise convolutional layer. For VGG, we compress the fully connected layers in addition to convolutions.

For the experiments on the CIFAR-10 dataset, we perform model training as described in He et al. He et al. (2016). We train all models for a total of 400 epochs at a batch size of 64, an initial learning rate of 0.1 with a cosine decay schedule Loshchilov & Hutter (2016). The weight decay is set to 0.0001. We run each experiment 5 times and report the median test accuracy. In first 200 epochs, the target rate of model increase from 0 to the target value which is related to the sparsity level we want to reach and the group size we have chosen. During this process, the dropout rate remained at 50%. In the remained epochs, the dropout rate ramping from 50% to 100% gradually.

Table A.1: Test accuracy of the ResNet-32 model on the CIFAR-10 dataset. Top rows: results from various sparse structures. (Baseline: baseline non-sparse model; $G = 16 \,/\, s = 1$: CC-Sparse model with group size $G = 16$, and $s = 1$; without group 1: sparse model with same sparse density as $G = 16 \,/\, s = 1$ but without the group structure; $G = 64 \,/\, s = 1$: CC-Sparse model with group size $G = 64$, and $s = 1$; without group 2: sparse model with same sparse density as $G = 64 \,/\, s = 1$ but without the group structure.) Bottom rows: results from the original TD study Gomez et al. (2018).

Model	Param Pruned (%)	Flops Pruned (%)	Top1 Accuracy (%)
Baseline	0.00	0.00	92.98
$G = 16 \,/\, s = 1$	93.51	93.15	90.56
Without Group 1	93.51	93.15	91.21
$G = 64 \,/\, s = 1$	97.40	95.50	88.39
Without Group 2	97.40	95.50	89.42
Original TDGomez et al. (2018)	94.00	94.00	88.80
Original TD	97.00	97.00	88.67
Original TD	98.00	98.00	88.70

A.2 EXCLUSIVE LASSO REGULARIZATION

Exclusive Lasso regularization encourages competition between components inside a group Zhou et al. (2010) through applying the following regularizer:

$$\ell(W) = \sum_{j=1}^{d} \left(\sum_{k=1}^{G} \left| W_k^j \right| \right)^2 \tag{A.1}$$

By using the ℓ_1 norm to combine the weights from the same group, which tends to give sparse solution, and ℓ_2 norm to combine different groups together, which tends to minimize the regularizer loss, the exclusive Lasso regularizer essentially encourages the weights inside a group to compete for non-zero weights positions Zhou et al. (2010). Such a property is desirable for models with CC-Sparsity. We performed experiments on combining the exclusive Lasso regularization with CC-Sparsity. When we use the exclusive Lasso regularizer alone for CC-Sparsity models training, we find that models generally perform poor. However, when we combine exclusive Lasso regularizer

Table A.2: Test accuracy of the VGG-16 model on the CIFAR-10 dataset. Top rows: results from various sparse structures. (Baseline: baseline non-sparse model; $G = 16 \,/\, s = 1$: CC-Sparse model with group size $G = 16$, and $s = 1$; without group 1: sparse model with the same sparse density as $G = 16 \,/\, s = 1$ but without the group structure; $G = 64 \,/\, s = 1$: CC-Sparse model with group size $G = 64$, and $s = 1$; and without group 2: sparse model with the same sparse density as $G = 64 \,/\, s = 1$ but without the group structure.) Bottom rows: comparison between CC-Sparsity and structured sparsity Li et al. (2016).

Model	Params Pruned (%)	Flops Pruned (%)	Top1 Accuracy (%)
Baseline	0.00	0.00	93.55
$G = 16 \,/\, s = 1$	93.68	93.21	92.55
Without Group 1	93.68	93.21	92.90
$G = 64 \,/\, s = 1$	98.36	97.87	90.48
Without Group 2	98.36	97.87	91.46
$G = 16 \,/\, s = 4$	74.93	74.58	93.43
VGG-16 Li et al. (2016))	64.00	34.20	93.40

Table A.3: Performance comparison between two sparsity configurations on the ResNet-18 model. CC-Sparsity (G=16 / s=2) outperforms structured sparsity by a large margin.

Method	Baseline Top1 Accuracy(%)	Flops Pruned (%)	Top1 Accuracy (%)	Accuracy Drop(%)
Ours	71.12	70.55	70.31	0.81
Dong et al. Dong et al. (2017)	69.98	34.6	66.33	3.65
He et al. He et al. (2018)	70.28	41.8	67.10	3.18

with the TD training, we observe a much smoother transition in the validation accuracy curve when training ramp reaches the last step, as shown in Fig. A.1, indicating better model convergence. We also observe that a better final model validation accuracy than a model trained with TD (with ℓ_2 norm regularizer) can be achieved through tuning exclusive Lasso regularization strength. As this paper is more about the testing of concept instead of achieving state of the art results, the exclusive Lasso regularizer is not included in the main text.

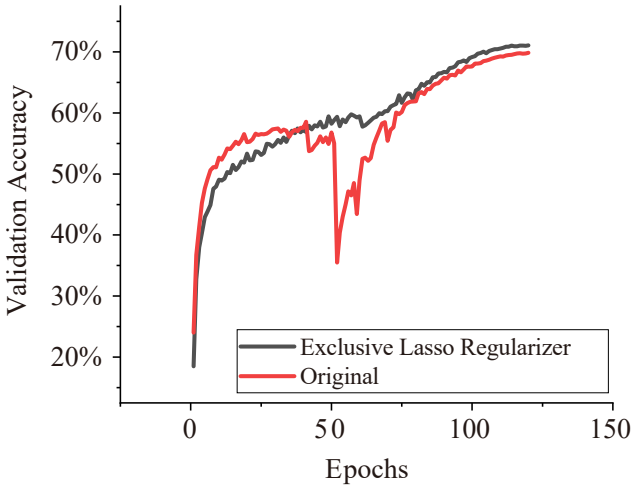


Figure A.1: The exclusive Lasso regularizer improves the TD model training.

Table A.4: Benchmark results on lightweight architectures

Model	Params	Flops	Top1 Accuracy%
MobileNet V1	4.2M	569M	70.60
MobileNet V1 (0.75)	2.6M	325M	68.30
MobileNet V1 (0.5)	1.3M	149M	63.70
MobileNet V2	3.4M	300M	72.00
MobileNet V2 (0.75)	2.61M	209M	69.80
MobileNet V2 (0.5)	1.95M	97M	65.40
IGCV3-D	3.5M	318M	72.20
IGCV3-D (0.7)	2.8M	210M	68.45
Condense (G=C=8)	2.9M	274M	71.00
ShuffleNet 1.5* (g = 3)	3.4M	292M	71.50
ShuffleNet 1* (g = 8)		140M	67.60
ShuffleNet 0.5* (shallow, g = 3)		40M	57.20
CI-Sparsity on MobilenetV2 (width=1, epoch=120, G=8, S=2)	2.2M	120M	69.46
CI-Sparsity on MobilenetV2 (width=1.4, epoch=120, G=8, S=2)	3.5M	222M	72.78
CI-Sparsity on MobilenetV2 (width=2.0, epoch=120, G=8, S=2)	5.2M	292M	73.65
CI-Sparsity on MobilenetV2 (width=1, epoch=400, G=8, S=4)	2.61M	180M	72.25
CI-Sparsity on MobilenetV2 (width=1, epoch=400, G=8, S=2)	2.19M	120M	71.21
CI-Sparsity on MobilenetV2 (width=1, epoch=400, G=8, S=1)	1.97M	90M	69.15