PDP: Parameter-free Differentiable Pruning is All You Need

Minsik Cho [1] **Saurabh Adya** [1] **Devang Naik** [1]

Abstract

In this paper, we propose an efficient yet effective train-time pruning scheme, Parameter-free Differentiable Pruning (PDP), which offers state-of-the-art qualities in model size, accuracy, and training cost. PDP uses a dynamic function of weights during training to generate soft pruning masks for the weights in a parameter-free manner for a given pruning target. While differentiable, the simplicity and efficiency of PDP make it universal enough to deliver state-of-the-art random/structured/channel pruning results on various vision models. For example, for MobileNet-v1, PDP can achieve 68.2% top-1 ImageNet1k accuracy at 86.6% sparsity, which is 1.7% higher accuracy than those from the state-of-the-art algorithms. PDP also improved the top-1 ImageNet1k accuracy of ResNet18 by over 3.6% and reduced the top-1 ImageNet1k accuracy of ResNet50 by 0.6% from the state-of-the-art.

1. Introduction

Deep neural networks (DNN) have shown human performance on complex cognitive tasks (Silver et al., 2018), but their deployment onto mobile/edge devices for enhanced user experience (i.e., reduced latency and improved privacy) is still challenging. Most such on-device DNN systems are heavily resource-constrained, thus requiring high power/compute/storage efficiency (Howard et al., 2017; Vasu et al., 2023; Wang et al., 2019; Wu et al., 2018).

Such efficiency can be accomplished by mixing and matching various techniques, such as designing efficient DNN architectures like MobileNet/MobileViT/ MobileOne (Mehta & Rastegari, 2022; Sandler et al., 2018; Vasu et al., 2023), distilling a complex DNN into a smaller architecture (Polino et al., 2018), quantizing/compressing the weights of DNNs (Cho et al., 2022; Han et al., 2016; J. Lee, 2021; Li et al., 2019; Park & Yoo, 2020; Zhao et al., 2019), and pruning near-zero weights (Kusupati et al., 2020; Liu et al., 2021; Peste et al., 2021; Sanh et al., 2020; Wortsman et al., 2019; Zafrir et al., 2021; Zhang et al., 2022; Zhu & Gupta, 2018). Also, pruning is known to be highly complementary to quantization/compression (Wang et al., 2020b) when optimizing a DNN model. However, pruning comes at the cost of degraded model accuracy, and the trade-off is not straightforward (Kusupati et al., 2020).

Hence, a desirable pruning algorithm should achieve high accuracy and accelerate inference for various types of networks without significant training overheads in costs and complexity. In this work, we propose a simple yet effective pruning technique, Parameter-free Differentiable Pruning or PDP, which uses a dynamic function of weights to generate soft pruning masks for the weights themselves. PDP requires neither additional learning parameters (Zhang et al., 2022) nor complicated training flows (Peste et al., 2021), yet offers precise control on the target sparsity level (Kusupati et al., 2020), while pushing the state-of-the-arts in random/structured/channel pruning.

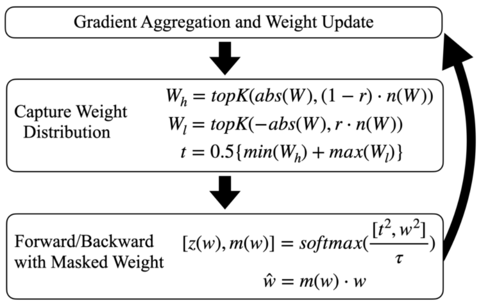
- PDP outperforms the state-of-the-art schemes on a variety of models and tasks by being differentiable and parameter-free without complex techniques.
- PDP offers a universal approach for efficient random/structured/channel pruning, while delivering a high-quality model optimization for a given pruning target.
- With a dynamic function of weights, PDP generates a soft pruning mask without training, and thus does not require gradient synchronization and SGD-update.

2. PDP: Parameter-free Differentiable Pruning

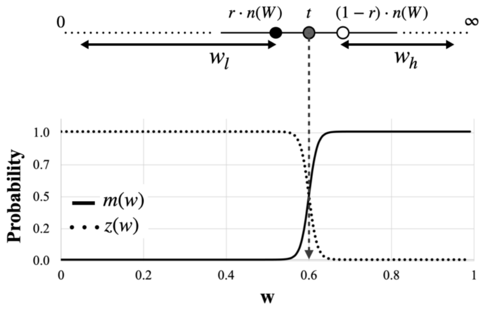
Complex pruning schemes do not always yield the best quality results, and their complexity and cost can make them impractical and difficult to use. The proposed Parameter-free Differentiable Pruning (PDP) is a highly effective and efficient scheme that generates soft pruning masks using a dynamic function of weights in a parameter-free and differentiable fashion. Since PDP is differentiable, the task loss can directly guide the pruning decision, offering an effective pruning solution. Simultaneously, being parameter-free, PDP can be fast and less intrusive to the existing training flow. Overall, PDP finds a weight distribution that is best

[1]Apple. Correspondence to: Minsik Cho <minsik@apple.com>.

Published at the Differentiable Almost Everything Workshop of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. July 2023. Copyright 2023 by the author(s).


Gradient Aggregation and Weight Update
Capture Weight Distribution
$W_h = topK(abs(W), (1-r)\cdot n(W))$
$W_l = topK(-abs(W), r\cdot n(W))$
$t = 0.5\{min(W_h) + max(W_l)\}$
Forward/Backward with Masked Weight
$[z(w), m(w)] = softmax(\frac{[t^2, w^2]}{\tau})$
$\hat{w} = m(w)\cdot w$

(a) PDP training flow



(b) Probability plot by $w \in R_{[0,1]}$ with $t = 0.6$

Figure 1. Computing $z(w), m(w)$ for the chances for Z and M with t for the equal chance to be in Z and M.

for task loss and pruning. Instead of having extra parameters, PDP indirectly influences the weight distribution for high-quality pruning. For example, if a weight w is destined to be pruned for some reason, instead of having a new parameter to denote **"to-prune"**, PDP lets SGD gradually make w itself smaller relatively against other parameters in the same entity, increasing its chance to be pruned over time. We will first discuss the benefits of PDP over existing differentiable pruning approaches in Section 2.1, present PDP in Section 2.2, then show the extension to structured and channel pruning in Section 2.3.

2.1. PDP Benefits

Learning/generating pruning masks with extra parameters allows the pruning decision to be driven by a task loss through back-propagation rather than the weight value itself (i.e., a hard mask will zero out the gradient of a pruned weight), but comes with the following issues.

- A pruning mask is (or derives from) a learnable parameter, increasing the trainable parameter count significantly and making the training process slow and complex (Elkerdawy et al., 2022; Romero et al., 2022; Sanh et al., 2020; Savarese et al., 2020; Zhang et al., 2022).
- A hard mask is approximated into a soft mask using a differentiable function, without guaranteeing the key properties of a pruning mask, such as the [0,1] value range or monotonicity (Ramakrishnan et al., 2020).

2.2. PDP Algorithm

To address the drawbacks of existing differentiable pruning algorithms, we propose PDP. A soft mask should ideally represent the chance of a weight w being in one of two symbolic states, "to-prune" (noted as Z) or "not-to-prune" (noted as M), without requiring extra parameters or expensive book-keeping. While the chance of w being in either state is not straightforward to compute, PDP generates a soft mask based on the fact that there exists an equal chance point for both states. Let us consider differentiable functions, $z, m : \mathbb{R}[0,\infty] \mapsto \mathbb{R}[0,1]$, to compute the chances of being in Z and M, which must satisfy the following conditions as a soft mask for magnitude-based pruning:

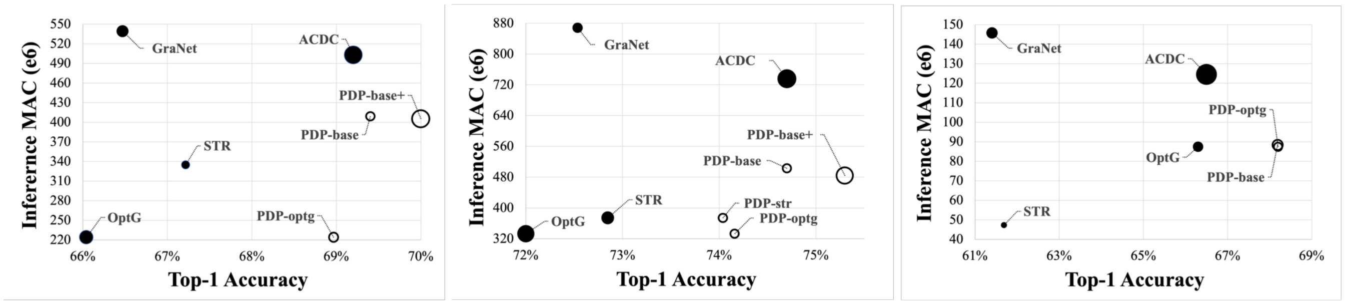
- $z(|a|) < z(|b|)$ for $|a| > |b|$: a weight with a smaller magnitude has a higher chance to be in Z.
- $m(|a|) > m(|b|)$ for $|a| > |b|$: a weight with a larger magnitude has a higher chance to be in M.
- $z(w) + m(w) = 1$ for any w: the total probability is 1.

$$z(w) = \begin{cases} 1 & \text{if } w = 0 \\ 0 & \text{if } |w| = \infty \\ \frac{1}{2} & \text{if } |w| = t \end{cases}$$

Then, by the monotonicity and continuity, there exists $t \in \mathbb{R}_{\geq 0}$ such that $z(t) = m(t) = 0.5$ (the equal chance for Z and M), which leads to the following boundary conditions on the left. Any function that satisfies these conditions can be used to compute $m(w)$ as a soft mask of w for train-time pruning. Let denote that $topK(X, k)$ is selecting the largest k elements from a matrix X, $abs(X)$ is an element-wise absolute operation, and $n(X)$ returns the element count. In PDP, we uniquely identify t for a given prune ratio $r \in [0, 1)$ for a layer with a weight matrix W as in Fig. 1 (a).

- The sparsity r for each W can be easily obtained by sorting the weights from the network by magnitude after a few epochs w.r.t the global target sparsity as a one-time task, or set by a user.
- Right after the SGD weight update, t is computed for the weights W in each layer or entity. The role of t is to abstract the current weight distribution of each layer/entity for pruning.
- During forward-pass, a soft mask, $m(w)$ for the weight w is computed and applied for the masked weight $\hat{w}$, which is differentiable. τ is the temperature parameter.
- Computing t and generating $\hat{w}$ repeat iteratively to adapt to the updated weight distribution.

Figure 1 (b) shows how the value of t is obtained in PDP and a soft mask is computed. Specifically, t is set to the value that is halfway between the largest pruned weight and the smallest unpruned weight when a hard mask is applied for a given sparsity ratio r. This ensures that each weight



(a) ResNet18 (sparsity: 85.5%) (b) ResNet50 (sparsity: 89.8%) (c) MobileNet-v1 (sparsity: 86.6%)

Figure 2. PDP-powered pruning (in white box markers) delivers the Pareto superiority to the other schemes (i.e., the top-bottom corner is the best trade-off) for ResNet18, ResNet50, and MobileNet-v1 on ImageNet1k. The size of markers indicates the relative training overheads. The detailed numbers are in Table 5 in Appendeix.

has an equal probability of being pruned or kept. As a result, PDP satisfies all the constraints for z and m. More details on PDP training flow are in Section C in Appendix.

PDP uses a dynamic function of t to generate soft pruning masks of W without the need for any extra trainable parameters. Instead, PDP lets the weights of the network adjust themselves such that the information that would otherwise be learned by the extra trainable parameters is instead fused into the weights themselves and their distribution. This is possible because each weight w is not only a coefficient in a layer, but also an indicator of the relative chance of that weight being pruned against the other weights in W. This relative chance is from the value of t in Figure 1 (b).

2.3. PDP for Structured and Channel Pruning

The simplicity and non-intrusive nature of PDP make it readily applicable to differentiable structured and channel pruning. As an example of structured pruning, we consider N:M pruning, where only N weights are kept out of every M consecutive weights. N:M pruning is attracting high research and industrial attention because top-of-the-line GPUs support 2:4 configuration (Jeff Pool, 2021). To apply PDP to N:M pruning, we apply it to every M consecutive weights of the layer, as if the layer were composed of many sublayers, each with M weights. Since N:M dictates the target sparsity, we can easily find the threshold t and generate the soft mask, as shown in Figure 1 (a).

Channel pruning is another type of structured pruning that can be easily applied to PDP with minor modifications. To do this, we first compute the L_2 norm of each channel in the layer, and then use these norm values (in place of the absolute values of the weights in Figure 1(a)) to compute a soft mask for each channel. Using the soft mask to prune all the corresponding weights in the channel will make the channel pruning process differentiable. For the illustrations on N:M and channel pruning usine PDP, please refer to Fig. 7 in Appendix.

3. Experimental Results

We compared **PDP** with state-of-the-art random, structured, and channel pruning schemes on various computer vision models. We used two x86 Linux nodes with 8 NVIDIA V100 GPUs on each in cloud. All cases were trained from scratch. More experimental results and the hyperparameters are in Section E and Table 3 in Appendix.

Random Pruning: We compared **PDP** with the latest prior arts, **STR** (Kusupati et al., 2020), **GMP** (Zhu & Gupta, 2018), **DNW** (Wortsman et al., 2019), **GraNet** (Liu et al., 2021), **OptG** (Zhang et al., 2022), and **ACDC** (Peste et al., 2021) on ResNet18, ResNet50, and MobileNet-v1 (He et al., 2016; Howard et al., 2017) with the ImageNet1k dataset (Deng et al., 2009). Since all of these schemes have been experimented only with ResNet50 and/or MobileNet-v1, we reproduced the pruning results in our controlled environment with the identical data augmentations by running the official implementations from the authors (Kusupati et al., 2020; Liu et al., 2021; Peste et al., 2021; Zhang et al., 2022) or verified implementations from the prior arts (Wortsman et al., 2019; Zhu & Gupta, 2018) as in Section F in Appendix. We measured the accuracies and Multiply-Accumulate Operation (MAC) during inference on each experiment with layer fusion (i.e., BatchNorm folding), and mainly focused on the high-sparsification cases. In our experiments with ImageNet1k, all layers are pruned.

Since each algorithm used a different number of epochs and showed results at different sparsity levels, **a)** we ran **STR** first to set the target sparsity levels for all the networks for fair comparisons, because all other schemes can control the sparsity level precisely, **b)** we trained ResNet18/50 for 100 epochs and MobileNet-v1 for 180 epochs following (Kusupati et al., 2020; Liu et al., 2021; Peste et al., 2021; Zhang et al., 2022) except **STR** (which diverged with more epochs for MobileNet-v1). For **PDP**, we fixed the target sparsity for each layer based on the global weight magnitude at the epoch 16 and started pruning at the rate of 1.5% of the target

Network	Method	Batch size	#epochs	N:M				avg GPU cost ($)
				2:4	4:8	1:4	2:8	
ResNet18	LNM	256	120	69.6	**70.2**	65.1	68.4	395
	PDP	1024	100	**70.2**	70.1	**68.7**	**69.1**	275
ResNet50	LNM	256	120	74.6	75.1	74.1	75.0	812
	PDP	1024	100	**75.9**	**75.8**	**75.0**	**75.3**	380

Table 1. Structured Pruning: **PDP** can be directly to do N:M pruning due to its generality. PDP delivers the superior results than the latest N:M pruning in (Zhou et al., 2021) at 46% less training cost.

Method	Batch size	#epochs	Top-1 (%)	MAC drop (%)
NISP	?	90	75.3	44.0
DCP	256	60	75.0	55.0
SCP	256	100	75.3	54.3
PDP	1024	100	**75.9**	54.9

[*] **SCP, DCP**, and **NISP** reported only ResNet50 results with MAC drop instead of sparsity. Hence, for **PDP**, we report the nearest MAC drop we obtained (54.9%) at 57% channel sparsity.

Table 2. Channel Pruning: the generality of **PDP** helps deliver the state-of-the-art results without modifications.

sparsity per epoch for all the experiments which correspond to $s = 16$ and $\epsilon = 0.015$ in Algorithm 1 in Appendix. For detailed experiment configurations, please refer to Table 3. Every experiment began with a randomly initialized model (i.e., no pre-trained model). For **PDP**, we had the following variants to show the value of **PDP** with the same training overhead or per-layer pruning budgets.

- **PDP-base** globally computes the target sparsity by $abs(W)$ at epoch 16 across all the layers.
- **PDP-base+** is the same as **PDP-base** yet with more epochs to match the GPU cost of **ACDC**.
- **PDP-str/optg** uses the per-layer sparsity from **STR/OptG** as input to normalize the MAC.

Our experimental results are highlighted in Fig. 2, where the size of circles indicates the relative training overhead due to pruning. Note that we used only one single node with 8 GPUs due to the limitation in the official implementations for **GraNet** and **OptG**, thus both have the advantage of not having the inter-node communication cost. Also, each approach imposes a different level of training-time overhead, mainly due to the various complexities of training flow and pruning itself as captured in Fig. 2. Overall results can be summarized as follows:

- **PDP** showed the best the model accuracy: **PDP-base** on ResNet18 delivered 69% Top-1 accuracy which is superior to other schemes but at higher MAC than only **STR** and **OptG**.
- **PDP** offered the better model accuracy for a given pruning target: With the custom sparsification target for each layer, **PDP-str/optg** demonstrated the 2-3% higher Top-1 accuracy at the same MAC, demonstrating the effectiveness of the proposed method.
- When we use the similar GPU budget for additional epochs with **ACDC** which is noted as **PDP-base+**, our method further improved the Top-1 accuracy from 69% to 69.5% for ResNet18 and from 74.7% to 75.3% for ResNet50 with slight fewer MACs.

Structured/Channel Pruning: We compared PDP-driven N:M pruning and channel pruning on the ImageNet1k dataset (Deng et al., 2009). For N:M pruning, we reproduced the **LNM** (Zhou et al., 2021) results using the public code base but without the color augmentation to keep the experimental environment normalized. For **PDP**, we simply reused the hyper-parameters and configurations as in Table 3 in our Appendix, and the top-1 accuracies by various N:M configs with ImageNet1k on ResNet18/50 are presented in Table 1. We can observe that **PDP** outperforms **LNM** on all the test cases, even with 4x larger batch size in 20 fewer epochs. **LNM** training cost is also much higher than **PDP** because of its costly weight regularization and complex back-propagation scheme.

For channel pruning, we compared **PDP** with **SCP** (Kang & Han, 2020a), **NISP** (Yu et al., 2018), and **DCP** (Zhuang et al., 2019). Note that **SCP** uses the β in BatchNorm to select the channels to prune (i.e., $beta \leq \epsilon$), hence applicable to limited types of networks only. We again reused the hyper-parameters and configurations as in Table 3 in our Appendix for **PDP**, and the top-1 accuracy with ImageNet1k on ResNet50 is reported in Table 2 where we can see that **PDP** shows superior performance for channel pruning.

4. Conclusion

We showed that a simple and universal pruning method PDP can yield the state-of-the-art random/structured/channel pruning quality on popular computer vision models. Our method requires no additional learning parameters, yet keeps the training flow simple and straightforward, making it a practical method for real-world scenarios.

References

Anwar, S., Hwang, K., and Sung, W. Structured pruning of deep convolutional neural networks. *ACM Journal on Emerging Technologies in Computing Systems*, 2017.

Bengio, Y., Léonard, N., and Courville, A. C. Estimating or propagating gradients through stochastic neurons for conditional computation. *CoRR*, 2013.

Cho, M., Alizadeh-Vahid, K., Adya, S., and Rastegari, M. DKM: differentiable k-means clustering layer for neural network compression. In *International Conference on Learning Representations*, 2022.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. ImageNet: A Large-Scale Hierarchical Image Database. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

Elkerdawy, S., Elhoushi, M., Zhang, H., and Ray, N. Fire together wire together: A dynamic pruning approach with self-supervised mask prediction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022.

Evci, U., Gale, T., Menick, J., Castro, P. S., and Elsen, E. Rigging the lottery: Making all tickets winners. In *International Conference on Machine Learning*, 2020.

Frankle, J. and Carbin, M. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2019.

Gale, T., Elsen, E., and Hooker, S. The state of sparsity in deep neural networks. *CoRR*, 2019.

Han, S., Pool, J., Tran, J., and Dally, W. J. Learning both weights and connections for efficient neural network. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, 2015.

Han, S., Mao, H., and Dally, W. J. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In *International Conference on Learning Representations*, 2016.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

He, Y., Zhang, X., and Sun, J. Channel pruning for accelerating very deep neural networks. In *The IEEE International Conference on Computer Vision*, Oct 2017.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

J. Lee, D. Kim, B. H. Network quantization with element-wise gradient scaling. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2021.

Jeff Pool, C. Y. Channel permutations for n:m sparsity. In *Advances in Neural Information Processing Systems*, 2021.

Kang, M. and Han, B. Operation-aware soft channel pruning using differentiable masks. In *International Conference on Machine Learning*, 2020a.

Kang, M. and Han, B. Operation-aware soft channel pruning using differentiable masks. In *International Conference on Machine Learning*, 2020b.

Kusupati, A., Ramanujan, V., Somani, R., Wortsman, M., Jain, P., Kakade, S., and Farhadi, A. Soft threshold weight reparameterization for learnable sparsity. In *International Conference on Machine Learning*, July 2020.

Lagunas, F., Charlaix, E., Sanh, V., and Rush, A. M. Block pruning for faster transformers. In *The Conference on Empirical Methods in Natural Language Processing*, 2021.

Li, H., Kadav, A., Durdanovic, I., Samet, H., and Graf, H. P. Pruning filters for efficient convnets. In *International Conference on Learning Representations*, 2017.

Li, Y., Dong, X., and Wang, W. Additive powers-of-two quantization: An efficient non-uniform discretization for neural networks. In *International Conference on Learning Representations*, 2019.

Liu, S., Chen, T., Chen, X., Atashgahi, Z., Yin, L., Kou, H., Shen, L., Pechenizkiy, M., Wang, Z., and Mocanu, D. C. Sparse training via boosting pruning plasticity with neuroregeneration. In *Advances in Neural Information Processing Systems*, 2021.

Liu, Z., Whatmough, P. N., Zhu, Y., and Mattina, M. S2ta: Exploiting structured sparsity for energy-efficient mobile cnn acceleration. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2022.

Mehta, S. and Rastegari, M. Mobilevit: Light-weight, general-purpose, and mobile-friendly vision transformer. In *International Conference on Learning Representations*, 2022.

Mishra, A. K., Latorre, J. A., Pool, J., Stosic, D., Stosic, D., Venkatesh, G., Yu, C., and Micikevicius, P. Accelerating sparse deep neural networks. *CoRR*, 2021.

Park, E. and Yoo, S. Profit: A novel training method for sub-4-bit mobilenet models. In *European Conference on Computer Vision*, 2020.

Peste, A., Iofinova, E., Vladu, A., and Alistarh, D. AC/DC: alternating compressed/decompressed training of deep neural networks. In *Advances in Neural Information Processing Systems*, 2021.

Polino, A., Pascanu, R., and Alistarh, D. Model compression via distillation and quantization. In *International Conference on Learning Representations*, 2018.

Ramakrishnan, R. K., Sari, E., and Nia, V. P. Differentiable mask for pruning convolutional and recurrent networks. In *International Conference on Computer and Robot Vision*, 2020.

Romero, D. W., Bruintjes, R., Tomczak, J. M., Bekkers, E. J., Hoogendoorn, M., and van Gemert, J. C. Flexconv: Continuous kernel convolutions with differentiable kernel sizes. 2022.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, June 2018.

Sanh, V., Wolf, T., and Rush, A. Movement pruning: Adaptive sparsity by fine-tuning. In *Advances in Neural Information Processing Systems*, 2020.

Savarese, P., Silva, H., and Maire, M. Winning the lottery with continuous sparsification. In *Advances in Neural Information Processing Systems*, 2020.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

Tanaka, H., Kunin, D., Yamins, D. L. K., and Ganguli, S. Pruning neural networks without any data by iteratively conserving synaptic flow. In *Advances in Neural Information Processing Systems*, 2020.

Vasu, P. K. A., Gabriel, J., Zhu, J., Tuzel, O., and Ranjan, A. An improved one millisecond mobile backbone. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2023.

Wang, C., Zhang, G., and Grosse, R. B. Picking winning tickets before training by preserving gradient flow. In *International Conference on Learning Representations*, 2020a.

Wang, K., Liu, Z., Lin, Y., Lin, J., and Han, S. Haq: Hardware-aware automated quantization with mixed precision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

Wang, Y., Lu, Y., and Blankevoort, T. Differentiable joint pruning and quantization for hardware efficiency. In *European Conference on Computer Vision*, 2020b.

Wortsman, M., Farhadi, A., and Rastegari, M. Discovering neural wirings. In *Advances in Neural Information Processing Systems*, 2019.

Wu, J., Wang, Y., Wu, Z., Wang, Z., Veeraraghavan, A., and Lin, Y. Deep k-means: Re-training and parameter sharing with harder cluster assignments for compressing deep convolutions. In *International Conference on Machine Learning*, 2018.

Yu, R., Li, A., Chen, C., Lai, J., Morariu, V. I., Han, X., Gao, M., Lin, C., and Davis, L. S. NISP: pruning networks using neuron importance score propagation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018.

Zafrir, O., Larey, A., Boudoukh, G., Shen, H., and Wasserblat, M. Prune once for all: Sparse pre-trained language models. In *Advances in Neural Information Processing Systems*, 2021.

Zhang, Y., Lin, M., Chen, M., Xu, Z., Chao, F., Shen, Y., Li, K., Wu, Y., and Ji, R. Optimizing gradient-driven criteria in network sparsity: Gradient is all you need. In *CoRR*, 2022.

Zhao, X., Wang, Y., Cai, X., Liu, C., and Zhang, L. Linear symmetric quantization of neural networks for low-precision integer hardware. In *International Conference on Learning Representations*, 2019.

Zhou, A., Ma, Y., Zhu, J., Liu, J., Zhang, Z., Yuan, K., Sun, W., and Li, H. Learning n:m fine-grained structured sparse neural networks from scratch. In *International Conference on Learning Representations*, 2021.

Zhou, H., Lan, J., Liu, R., and Yosinski, J. Deconstructing lottery tickets: Zeros, signs, and the supermask. In *Advances in Neural Information Processing Systems*, 2019.

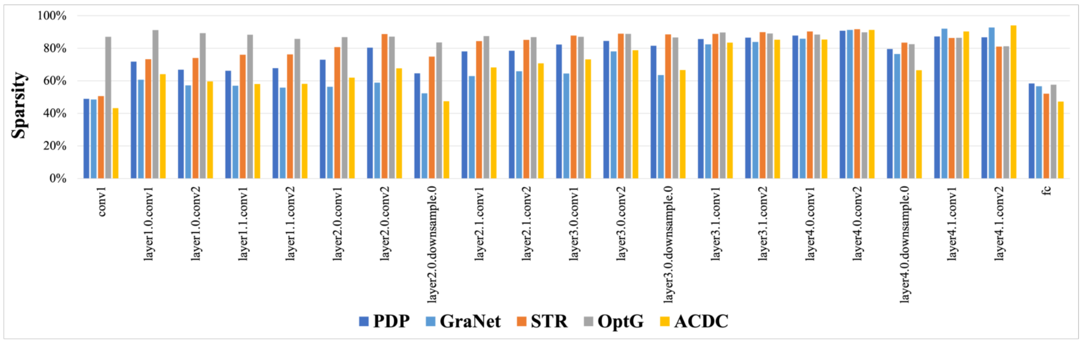
Zhu, M. and Gupta, S. To prune, or not to prune: Exploring the efficacy of pruning for model compression. In *Workshop, International Conference on Learning Representations*, 2018.

Zhuang, Z., Tan, M., Zhuang, B., Liu, J., Guo, Y., Wu, Q., Huang, J., and Zhu, J. Discrimination-aware channel pruning for deep neural networks. In *Advances in Neural Information Processing Systems*, 2019.

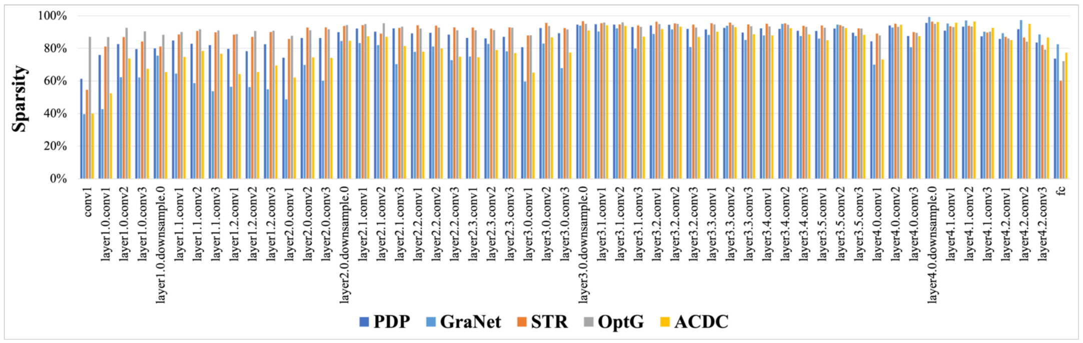
Network	Method	Batch size	#epochs	#wu	#GPUs	#Nodes	Main optimizer, scheduler	Mask optimizer, scheduler	other params
	CS	128	85x5	0	1	1	SGD 0.9 0.1, multi-step 56,75 by 0.1	same as the main	init value: -1
ResNet20	PDP	128	120	5	1	1	SGD 0.9 1e-4, cosine 0.4	n/a	$\tau : 1e^{-4}, \epsilon : 0.015$
	GraNet	128	100	5	8	1	SGD 0.9 1e-4 (0.0 for BN), cosine 0.1	n/a	init density: 0.5
	GMP	128	100	5	16	2	SGD 0.875 1e-5 , cosine 0.256	same as the main	
ResNet18	STR	256	100	5	16	2	SGD 0.875 2.251757813e-5 , cosine 0.256	same as the main	init value: -3200
	OptG	256	100	5	16	2	SGD 0.9 1e-4, cosine 0.1	SGD 0.9 0, cosine 0.1	$\beta : 1.0$
	ACDC	256	100	5	8	1	SGD 0.875 3.051757813e-4, cosine 0.256	n/a	8 alterations
	PDP-base/str/optg	1024	100	5	16	2	SGD 0.9 1.4e-5, cosine 1.8	n/a	$\tau : 1e^{-4}, \epsilon : 0.015$
	PDP-base+	1024	200	5	16	2	SGD 0.9 1.4e-5, cosine 1.8	n/a	$\tau : 1e^{-4}, \epsilon : 0.015$
	GraNet	128	100	5	8	1	SGD 0.9 1e-4 (0.0 for BN), cosine 0.1	n/a	init density: 0.5
	GMP	128	100	5	16	2	SGD 0.875 1e-5 , cosine 0.256	same as the main	
ResNet50	STR	256	100	5	16	2	SGD 0.875 2.251757813e-5, cosine 0.256	same as the main	init value: -3200
	OptG	256	100	5	16	2	SGD 0.9 1e-4, cosine 0.1	SGD 0.9 0, cosine 0.1	$\beta : 1.0$
	ACDC	256	100	5	8	1	SGD 0.875 3.051757813e-4, cosine 0.256	n/a	8 alterations
	PDP-base/str/optg	1024	100	5	16	2	SGD 0.9 1.4e-5, cosine 1.8	n/a	$\tau : 1e^{-4}, \epsilon : 0.015$
	PDP-base+	1024	200	5	16	2	SGD 0.9 1.4e-5, cosine 1.8	n/a	$\tau : 1e^{-4}, \epsilon : 0.015$
	GraNet	128	180	5	8	1	SGD 0.9 1e-4 (0.0 for BN), cosine 0.1	n/a	init density: 0.5
MobileNet-v1	STR	256	100	5	16	2	SGD 0.875 3.751757813e-5, cosine 0.256	same as the main	init value: -12800
	OptG	256	180	5	16	2	SGD 0.9 4e-4, cosine 0.1	SGD 0.9 0, cosine 0.1	$\beta : 1.0$
	ACDC	256	180	5	8	1	SGD 0.875 3.051757813e-4, cosine 0.256	n/a	8 alterations
	PDP-base/str/optg	1024	180	5	16	2	SGD 0.9 1.4e-5, cosine 1.8	n/a	$\tau : 1e^{-4}, \epsilon : 0.015$
	GraNet	128	180	5	8	1	SGD 0.9 1e-4 (0.0 for BN), cosine 0.1	n/a	init density: 0.5
MobileNet-v2	STR	256	100	5	16	2	SGD 0.875 3.751757813e-5, cosine 0.256	same as the main	init value: -12800
	OptG	256	180	5	16	2	SGD 0.9 4-e4, cosine 0.05	SGD 0.9 0, cosine 0.05	$\beta : 1.0$
	ACDC	256	180	5	8	1	SGD 0.875 3.051757813e-4, cosine 0.256	n/a	8 alterations
	PDP-base/str/optg	1024	180	5	16	2	SGD 0.9 8e-6, cosine 0.8	n/a	$\tau : 1e^{-4}, \epsilon : 0.015$

SGD: momentum, weight decay, cosine: learning_rate, AdamW: epsilon, multiplicative: learning rate, gamma, #wu: the number of warm-up epochs.

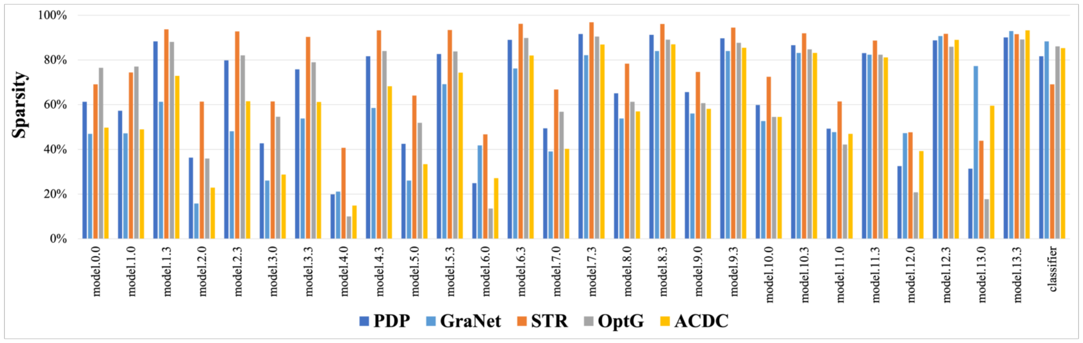
Table 3. The hyper-parameters in Sections 2 and 3.



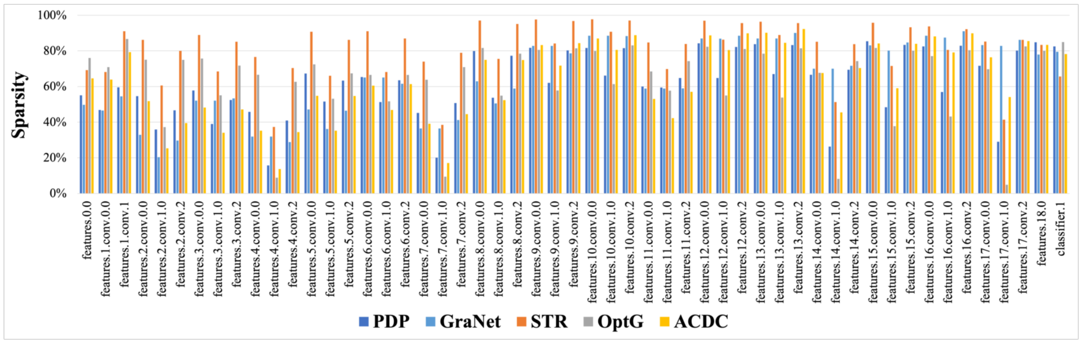
(a) ResNet18



(b) ResNet50

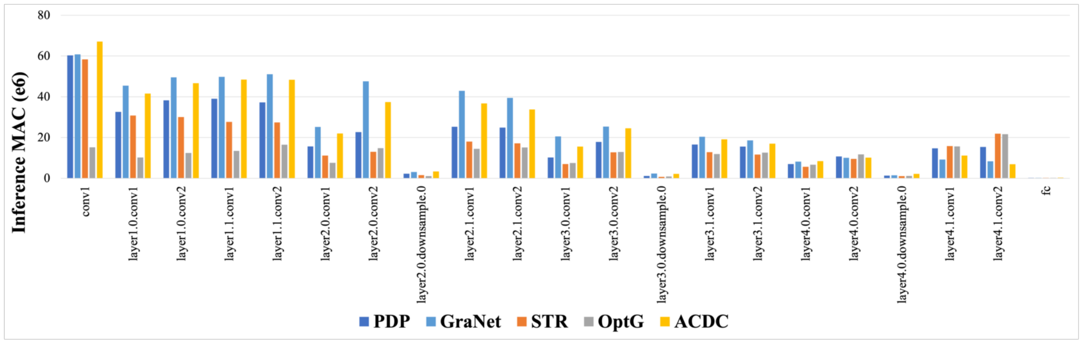


(c) MobileNet-v1

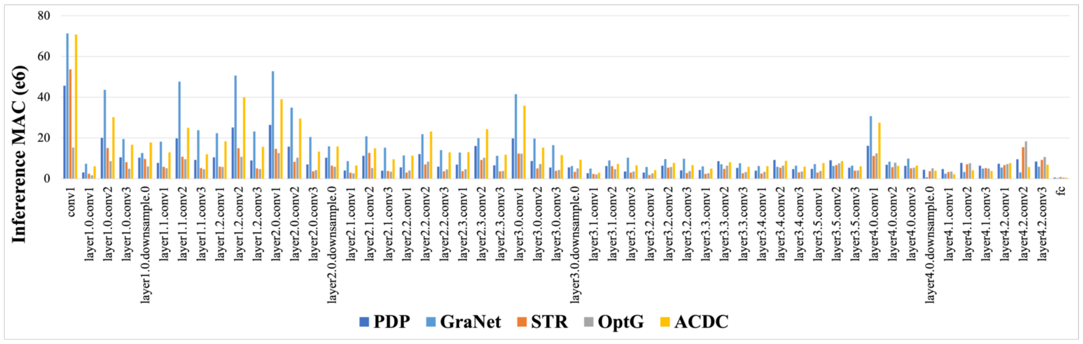


(d) MobileNet-v2

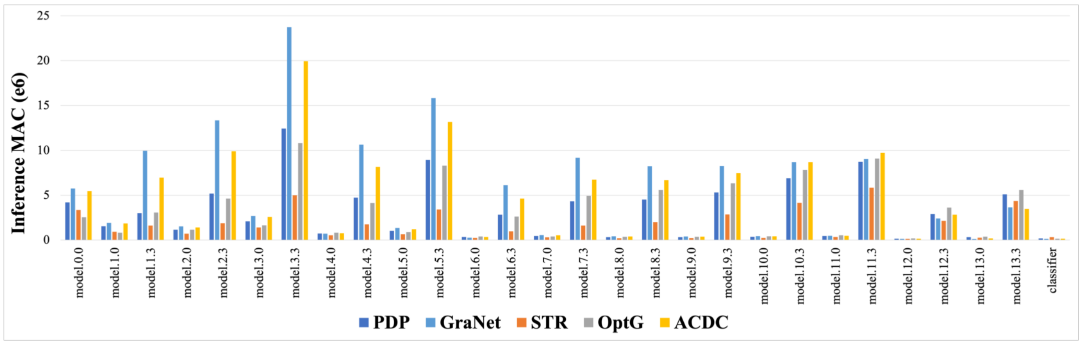
Figure 3. Layer-wise sparsity allocation from the experiments in Table 2.



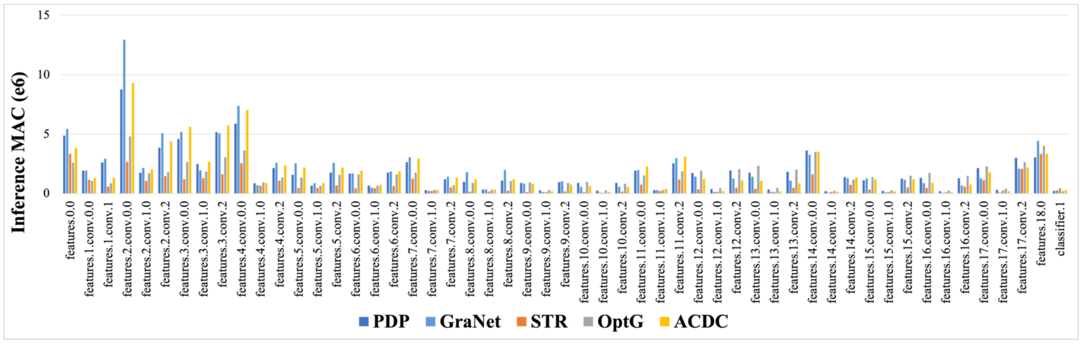
(a) ResNet18



(b) ResNet50



(c) MobileNet-v1



(d) MobileNet-v2

Figure 4. Layer-wise Inference MAC distribution from the experiments in Table 2.

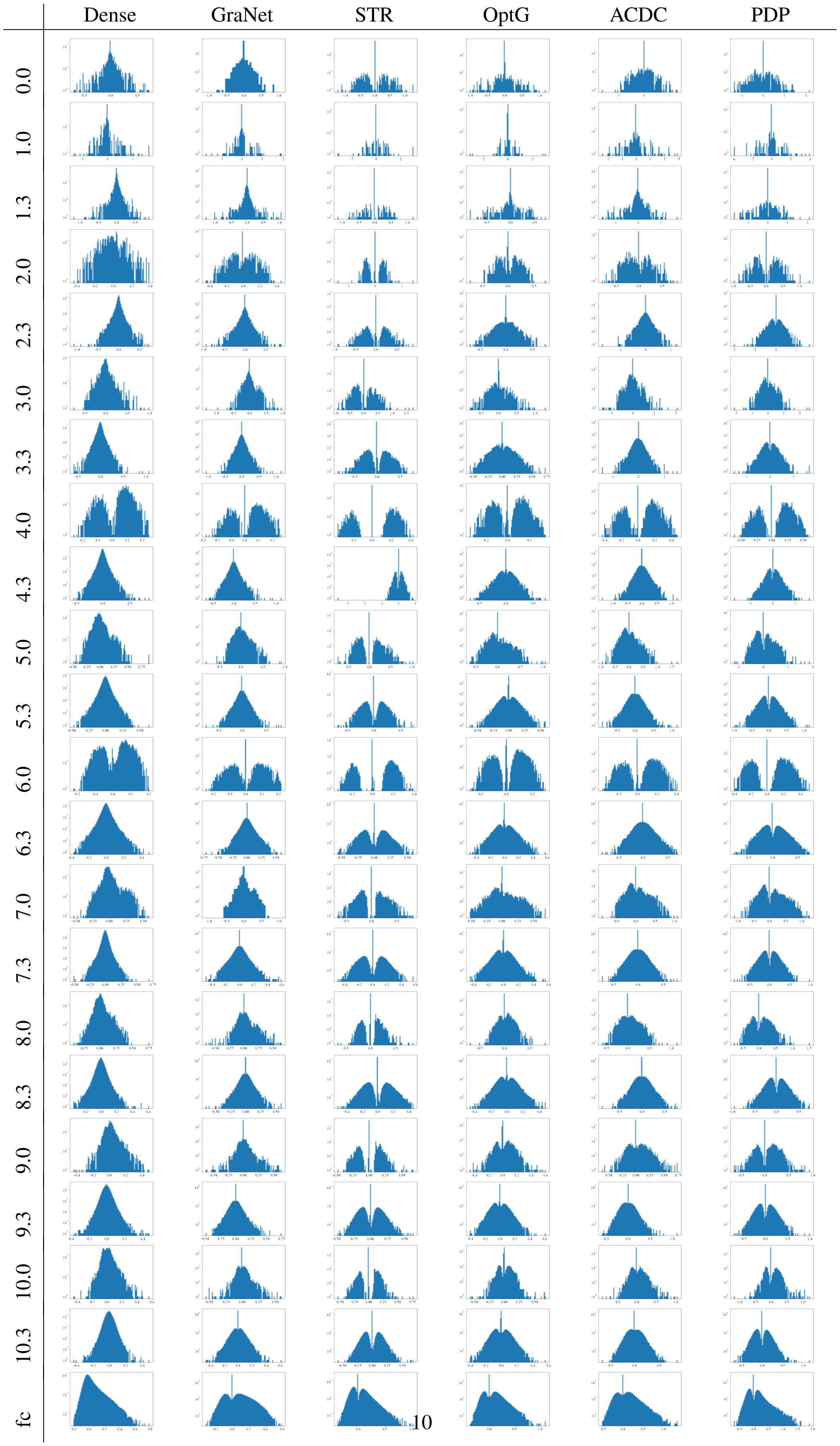


Table 4. The weight histograms in log scale for MobileNet-v1 in Table 2.

A. Training Configurations and Hyper-parameters

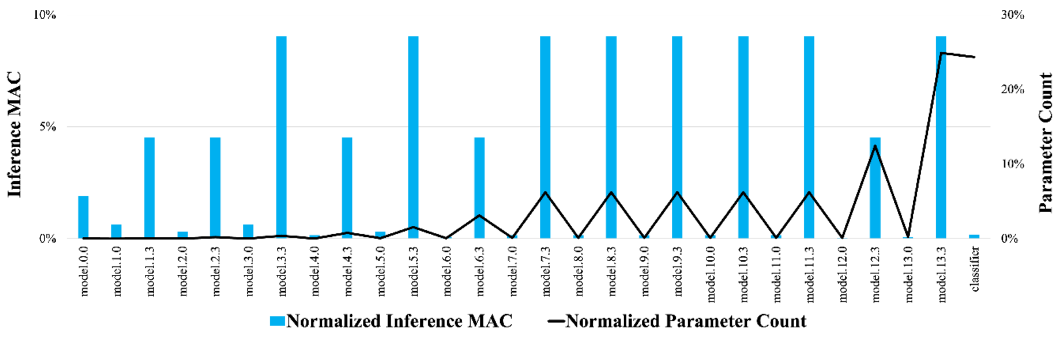
Since some techniques in Sections 2 and 3 require extra training parameters and pruning scheduling as shown in Table **??**, we disclose the training configurations and hyper-parameters we found the best in Table 3.

B. Trade-off in Pruning

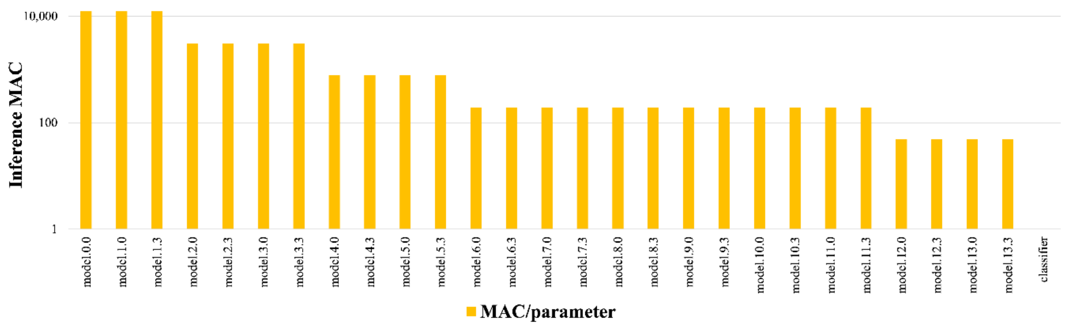
Pruning for DNN requires exploring a good trade-off between model accuracy and inference latency under a given pruning target. Such a challenge can be elaborated with the MobiletNet-v1 Dense case in Fig. 5 where the following observations can be made:

- The earlier layers have significantly fewer parameters than the later layers while still having comparable inference MACs as shown in (a). For example, the final classifier, which is a linear layer, has the lowest inference MAC but the 2nd largest parameters.
- When per-parameter inference MAC is computed as in (b) (which is in log-scale), we can easily see that the parameters in the earlier layers get much more involved in the inference than those in the later layers. For example, the MAC-per-parameter for the last classifier is only 1.

Then, with a given pruning target, one pruning scheme can favor heavily pruning the classifier, as it is easier to hit the target without affecting model accuracy much (i.e., each parameter shows up only once in the forward pass), but this would fail to reduce the inference MAC enough. Then, the other scheme may favor aggressively pruning the earlier layers to significantly minimize the inference latency at a much greater risk of degrading the model accuracy. Therefore, it is critical to find a good balance between accuracy and inference speed. According to our experimental results, PDP can accomplish such a balance using differential pruning w.r.t. the task loss. Such trade-off can be optimized differently depending on whether a particular sparsity pattern or structure is enforced.



(a) Normalized inference MAC and parameter count for each layer.



(b) The inference MAC per parameter for each layer.

Figure 5. Layer-wise Inference MAC and Parameters from the MobileNet-v1 Dense case in Table 2.

Network Sparsity	Method	Top-1 (%)	GPU$^{\$}$ cost($)	MAC ($\times e6$)	Network Sparsity	Method	Top-1 (%)	GPU$^{\$}$ cost ($)	MAC ($\times e6$)
ResNet18 85.5%	Dense	69.8	167	1814.1	ResNet50 89.8%	Dense	76.1	248	4089.2
	GMP	65.2	217	263.5		GMP	73.6	483	419.0
	DNW	64.4	206	263.5		DNW	70.7	466	419.0
	GraNet*	66.0	198	539.6		GraNet*	72.5	321	868.0
	STR	66.7	171	334.6		STR	72.8	417	373.7
	OptG*	65.5	277	223.7		OptG*	72.1	591	333.0
	ACDC	68.7	356	502.8		ACDC	74.7	635	735.6
	PDP-base	69.0	169	408.6		PDP-base	74.7	325	502.8
	PDP-base+	**69.5**	336	405.1		PDP-base+	**75.3**	610	483.0
	PDP-str	68.6	169	334.7		PDP-str	74.0	325	373.7
	PDP-optg	68.5	174	223.7		PDP-optg	74.2	332	332.9
MobileNet-v1 86.6%	Dense	70.9	277	568.7	MobileNetv-2 80.2%	Dense	71.9	297	300.8
	GraNet*	61.4	367	145.7		GraNet*	56.3	439	103.4
	STR	61.7	176	47.2		STR	60.0	285	40.6
	OptG*	66.3	340	87.4		OptG*	65.4	545	76.8
	ACDC	66.5	641	124.5		ACDC	64.1	812	93.9
	PDP-base	**68.2**	281	88.3		PDP-base	**66.8**	354	95.3
	PDP-str	65.3	307	47.2		PDP-str	60.7	307	40.6
	PDP-optg	68.2	297	87.3		PDP-optg	66.5	343	76.6

$^{\$}$ the GPU cost ($) is based on a commercial cloud spot instance pricing.
* used only one with 8 GPUs due to the limitations in the public code.

Table 5. PDP compared with other unstructured pruning algorithms on ImageNet1K shows the best trade-off among accuracy, inference MAC, and training overheads. More results are available in Section E in Appendix.

Random Pruning: Unstructured schemes make individual and independent pruning decision for each weight to maximize the flexibility and minimize the accuracy degradation. Simple and gradual/iterative pruning based on the weight magnitude has been studied extensively (Frankle & Carbin, 2019; Gale et al., 2019; Han et al., 2015; Zhu & Gupta, 2018). In these schemes, once a weight is pruned, it does not have the *second* chance to become unpruned and improve the model quality. To address such challenges, RigL (Evci et al., 2020) proposes to grow a sparse network by reallocating the removed weights based on their dense gradients. Applying brain-inspired neurogeneration (i.e., unpruning some weights based on gradients) and leveraging pruning plasticity is proposed (Liu et al., 2021). Altering the phase of dense and sparse training to accomplish co-training of sparse and dense models is studied, which results in good model accuracies on vision tasks (Peste et al., 2021). Unlike other magnitude-driven pruning, supermask training (Zhou et al., 2019) integrated with gradient-driven sparsity is proposed in (Zhang et al., 2022), where accumulated gradients are used to generate binary masks and straight-through estimator (Bengio et al., 2013) is used for backward propagation. Based on the lottery hypothesis (Frankle & Carbin, 2019), pruning in one-shot with heuristics (Tanaka et al., 2020) or gradient-driven metrics (Wang et al., 2020a) is explored.

Structured/Channel Pruning: Unstructured pruning limits inference latency speedup as it suffers from poor memory access performance, and does not fit well on parallel computation (Anwar et al., 2017; Liu et al., 2022). Recent research extends unstructured pruning by imposing a particular sparsity pattern during pruning at the cost of lower model predictive power, but increases the hardware performance during inference. One popular and effective form of structured pruning is channel pruning, where some channels with negligible effects on the model accuracy are discarded (He et al., 2017; Kang & Han, 2020b; Li et al., 2017). Using regularization to prune weights in a block is proposed in (Lagunas et al., 2021). (Kang & Han, 2020a) leverages the β in BatchNorm to select the channels to prune (i.e., $beta \leq \epsilon$) with ReLU assumed, which limits its applicability to wider set of DNNs. N:M pruning enforces that there are N non-zero weights out of every consecutive M weights (Zhou et al., 2021) which enables a compact memory layout and efficient inferences on hardware (Jeff Pool, 2021; Mishra et al., 2021; Zhou et al., 2021). A non-differentiable method for N:M pruning with complex back-prorogation based on STE is presented in (Zhou et al., 2021).

Algorithm 1 Training flow for PDP

```
1: procedure TRAIN($\epsilon, s, r, W = [W_0, W_1, ...]$)
2:     for epoch $e$ in $[0, 1, 2, s)$ do
3:         for each mini-batch do
4:             forward with $[W_0, W_2, ...]$
5:             backward-pass and update $W$
6:         end for
7:     end for
8:     $W_p = topK(-abs(W), r \cdot n(W))$
9:     $[r_0, r_1, ...] = [\frac{n(W_p \cap W_0)}{n(W_0)}, \frac{n(W_p \cap W_1)}{n(W_1)}, ...]$
10:    for epoch $e$ in $[s, s+1, s+2, ...]$ do
11:        $[\hat{r_0}, \hat{r_1}, ...] = min(1, \epsilon \cdot (e - s)) \cdot [r_0, r_1, ...]$
12:        for each mini-batch do
13:            for $i \in \{0, 1, ...\}$ do
14:                $W_h = topK(abs(W_i), (1 - \hat{r_i}) \cdot n(W_i))$
15:                $W_l = topK(-abs(W_i), \hat{r_i} \cdot n(W)_i)$
16:                $t_i = 0.5\{min(W_h) + max(W_l)\}$
17:            end for
18:            for $i \in \{0, 1, ...\}$ do
19:                $[Z_i, M_i] = softmax(\frac{[t_i^2 \mathbb{J}, W_i \circ W_i]}{\tau})$ //element-wise
20:                $\hat{W}_i$ = $M_i \circ W_i$ //element-wise
21:                forward-pass with $\hat{W}_i$
22:            end for
23:            backward-pass and update $W$
24:        end for
25:    end for
26:    $W_i = \lfloor M_i \rceil \circ W_i, \forall i \in \{0, 1, ..\}$
27: end procedure
```

C. PDP Algorithm and Training Flow

In order to obtain t in Fig 1, PDP needs a target pruning ratio r. The pruning ratio can be computed by selecting the top weights with larger magnitudes across all the layers and then instantly convert the selections into the per-layer ratios. Another way is to handcraft per-layer ratios, or reuse an existing configuration. Also, PDP is using the softmax operation which makes the *softness* concentrated over the weights around the t (as shown in Fig. 1). Hence, we gradually increase the target pruning ratio from 0 to r so that all low magnitude weights in the pruning range have a chance to use a soft-mask and settle down smoothly. For that purpose, we introduce a scaling step ϵ to let each weight have opportunities to leverage a soft-mask at least once, which leads to the training flow in Algorithm 1.

In lines 2-7, a normal training is performed for the first s epochs. Then, in lines 8 and 9, the per-layer target pruning ratio is computed by selecting the bottom $r \cdot n(W)$ weights globally in terms of the magnitude. Then, in the remaining epochs, we use PDP to generate soft-masks as in the line 15, while gradually increasing the target ratio as in lines 10 and 11. The updated weight distribution is captured by updating t_i as in the line 16 for all weight matrices. Once the entire training is over, we binarize the last mask for each weight to output the fully pruned weight for inference as in the line 26.

Overall, the average runtime complexity of PDP is $O(W)$, as we only need to exercise $topK$ algorithm (i.e, sorintg W is not necessary).

D. Ablation Study: Hyper-Parameter τ search

In the current PDP implementation, we use a global τ to control the level of softness in the pruning mask. Therefore, the selection of τ affects the model predictive power and should be carefully tuned. In order to explore the methodology for the τ search, we tried various values for MobileNet-v2 training, and the results are plotted in Fig. 6. The selection of τ affects the model predictive power as shown in Fig. 6 where there appears to be an optimal τ. For examples of MobileNet-v2, $\tau = 1e-4$ is the best value and is used for all the experiments in Section 3.

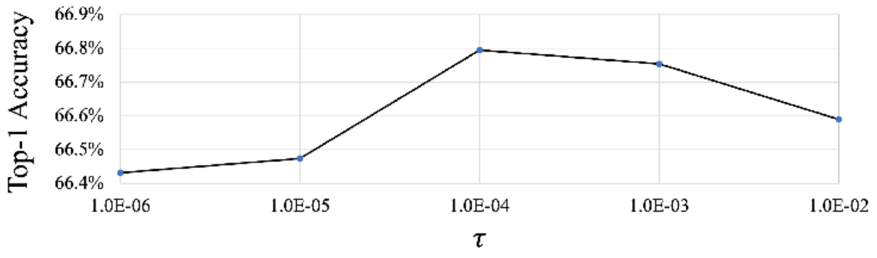


Figure 6. MobileNet-v2 with varying τ values.

Since, Fig. 6 shows a concave curve, one could use a binary search to find the best τ values w.r.t. the top-1 accuracy. Also, it could be possible to cast τ as a learnable parameter for each layer or apply some scheduling to improve the model accuracy further (as future work), but still both approaches need an excellent initial point which can be found using a binary search technique.

E. Additional Result for Section 3.

Different approaches made different sparsity allocations per the characteristics of the algorithm for a given pruning target, which results in complex trade-offs between model accuracy and inference speed. We report the detailed sparsity and inference MAC break-down for each layer in Fig. 3 and Fig. 4 on ImageNet1k and summarize our observations as follows:

- **OptG** prunes the early convolution layers quite aggressively in ResNet18 and ResNet50, which leads to very low inference MACs as shown in Fig. 2 (a) and (b), yet at the cost of the worse Top-1 accuracy. For example, the inference MAC of ResNet18 from **OptG** is more than 2x less than that from **ACDC**,
- Interestingly, **STR** becomes aggressive in pruning the early convolution layers in MobileNet-v1/v2, while **OptG** does not expose such behavior to MobileNet-v1/v2 (unlike it did for ResNet18/50). Such characteristics also favor the low inference latency over the model accuracy. Also, **STR** tends not to prune the last linear layer much as discussed in (Kusupati et al., 2020).
- Unlike **OptG** and **STR**, **ACDC** does not prune the early convolution layers much for the tested networks, but prunes somewhat actively for the late convolution and linear layers, which leads to high model accuracies at the cost of higher inference latencies.
- **PDP** is somewhat between **STR** and **ACDC** and modest across all layers in pruning allocation for all the networks, leading to superior accuracy and inference trade-offs. For example, the layers model.13.3 of MobileNet-v1 and features.17.conv.1.0 of MobileNet-v2 have the most difference among algorithms, and **PDP** is modest in pruning these two layers.
- **OptG** has very low inference MACs on the earlier layers of ResNet18 and ResNet50 due to its aggressive pruning on these as seen in Fig. 3 (a) and (b), which leads to the extremely low inference latencies as shown in Fig. 2 (a) and (b).
- **GraNet** tends to prune the earlier layers less but the later layers more in general which explains why **GraNet** shows the highest inference MACs in Fig. 2.

Table. 4 shows the pruned weight histograms of MobileNet-v1 from Table 2. We can observe that each algorithm affects the weight distribution in a slightly different way.

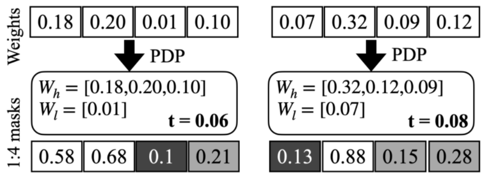
- **STR** prefers to split the distribution more widely than others. For the example of the layer 5.0, **STR** clearly separated the positive and negative weights with a wide gap centered at the zero, while others sis not, except **PDP** created a slight dip around the zero to create mild separation.

Network	Method	Sparsity (%)			
		80	70	60	50
MobileNet-v1	PDP	69.5	71.0	71.6	71.9
	OptG	68.1	69.1	69.6	69.7
	ACDC	68.5	69.9	70.9	71.4
ResNet-18	PDP	69.8	70.8	71.0	71.3
	ACDC	69.4	70.3	70.6	70.8

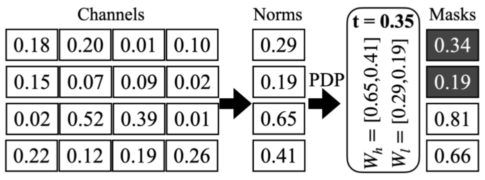
Table 6. Top-1 accuracy with ImageNet1k: **PDP** outperforms other schemes with various pruning rates.

Network	Method	Validation dataset	Sparsity (%)			
			80	70	60	50
Bert	PDP	matched	83.7	84.0	84.3	84.7
		mismatched	83.4	83.8	84.4	84.5
	OptG	matched	80.3	80.7	81.3	81.2
		mismatched	80.1	80.7	80.5	81.0

Table 7. Accuracies with MNLI benchmark: **PDP** maintains the similar accuracy lead over other schemes.



(a) 1:4 pruning (75% sparsity)



(b) Channel pruning (50% sparsity)

Figure 7. PDP is simple and universal enough to be applied directly to structured and channel pruning.

- **PDP** tends to spread out the sparsified weight distributions more than others. For the example of the fc layer, the weights from **PDP** range from -0.5 to 2.0, while those from others are from -0.5 to 1.5. On the other hand, **GraNet** tends to keep the weight distributions tight.

We also experimented with varying pruning rates for **PDP, OptG** and **ACDC** for MobiletNet-v1 and ResNet-18 with ImageNet1k, and Bert with MNLI benchmark under the same configurations as in Section 3. Overall, all tested schemes delivered higher accuracy with lower pruning rate, yet we can observe that **PDP** keeps its superiority to other schemes over all the tested pruning rates.

F. Code References

- **Dense** https://pytorch.org/vision/stable/index.html
- **GradNet** https://github.com/VITA-Group/GraNet
- **OptG** https://github.com/zyxxmu/OptG
- **ACDC** https://github.com/IST-DASLab/ACDC
- **STR** https://github.com/RAIVNLab/STR
- **GMP** https://github.com/RAIVNLab/STR
- **DNW** https://github.com/RAIVNLab/STR
- **CS** https://github.com/lolemacs/continuous-sparsification
- **LNM** https://github.com/NM-sparsity/NM-sparsity