Trainability Preserving Neural Pruning (TPP)

ICLR 2023

Huan Wang[1]

Yun Fu[1,2]

[1]Northeastern University, Boston, MA [2]AInnovation Labs, Inc

Code: https://github.com/MingSun-Tse/TPP

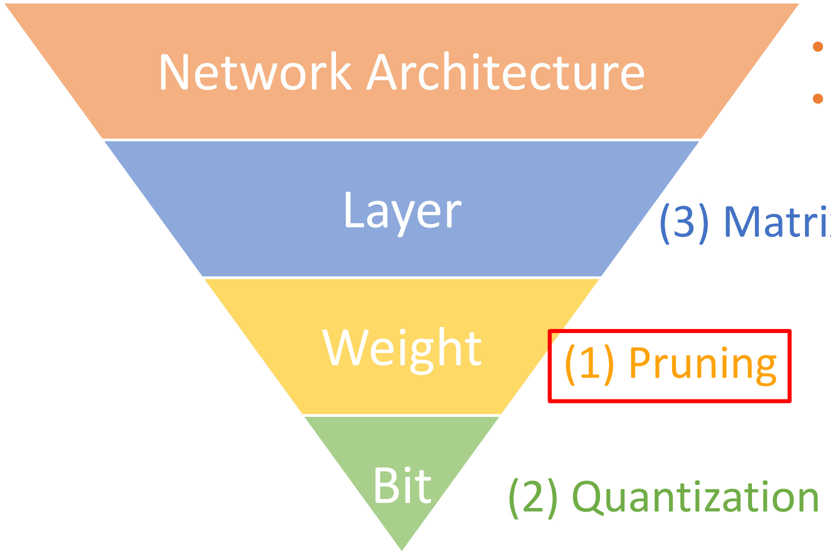
Categories of Model Compression / Efficient Deep Learning

Methods fall into 5 groups: (1) Pruning (2) Quantization (3) Low-rank decomposition (4) Knowledge distillation (5) Compact architecture design or search (AutoML: NAS, HPO)

large

granularity

small



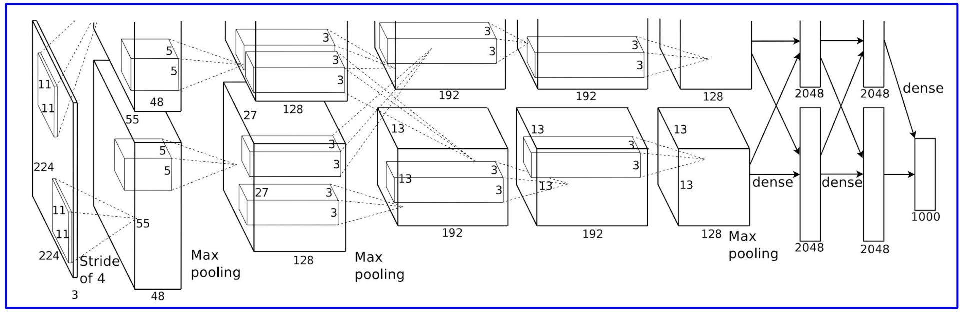
- (5) Hand-crafting (e.g., SqueezeNet[1], MobileNet[2], ShuffleNet[3])
- (5) Neural Architecture Searching (NAS) (e.g., EfficientNet [4])
- (4) Knowledge Distillation

(3) Matrix Decomposition

(1) Pruning

(2) Quantization

Hierarchical Redundancy of DNNs

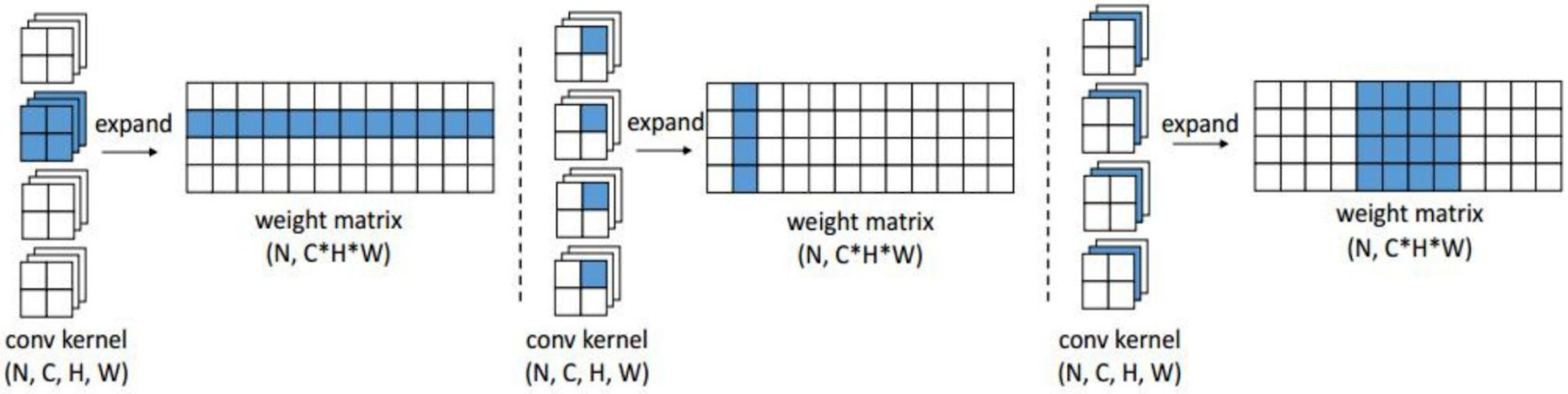


[AlexNet, 2012, NIPS]

[1] Iandola, Forrest N., et al. "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size." arXiv preprint arXiv:1602.07360 (2016).
[2] Howard, Andrew G., et al. "Mobilenets: Efficient convolutional neural networks for mobile vision applications." arXiv preprint arXiv:1704.04861 (2017).
[3] Zhang, Xiangyu, et al. "ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices." (2017).
[4] Tan, Mingxing, and Quoc V. Le. "Efficientnet: Rethinking model scaling for convolutional neural networks." In ICML (2019).

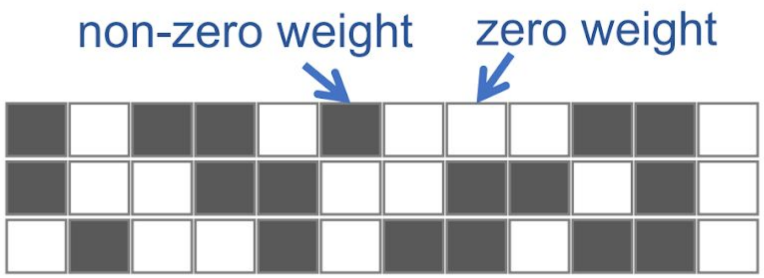
Structured pruning (hardware-friendly)

- **Im2col**: Convolutional kernels are expanded into weight matrices during convolution to leverage BLAS libraries.
- Popularized by deep learning framework Caffe.



(a) filter sparsity/row sparsity (b) shape sparsity/column sparsity (c) channel sparsity

Structured pruning



Unstructured pruning: much compression, but little speedup[1]

Regularity (Constraint)-Performance: It's all about *trade-off*!

You can choose anything to prune, depending on your specific need:

- Compression (storage reduction): element-wise pruning (unstructured pruning)
- Speedup: weight group (two choices: filter/row/channel pruning, column/shape-wise pruning)

[1] Wen et al., "Learning structured sparsity in deep neural networks", In NeurIPS, 2016

Our method is filter pruning (structured pruning)!

- Neural network pruning
- Trainability preserving

--- Next, talk about a bit about the background of why trainability matters ---

What is Trainability?

"Trainability, by its name, means the ability (easiness) of training a neural network."

-- Why is the State of Neural Network Pruning so Confusing?
On the Fairness, Comparison Setup, and Trainability in Network Pruning
https://arxiv.org/abs/2301.05219

≈ optimization speed

A key argument as the basis of our paper is that, **network pruning damages trainability, slowing down the optimization in the retraining stage**.

Basis of our ICLR'23 TPP Paper

Why is the State of Neural Network Pruning so Confusing?

On the Fairness, Comparison Setup, and <u>Trainability</u> in Network Pruning

Huan Wang

Can Qin

Yue Bai

Yun Fu

Northeastern University, Boston, MA

Arxiv: https://arxiv.org/abs/2301.05219

Q1: How much progress you expect in the past 6 years for filter/channel pruning?

ResNet50 on ImageNet, 2x - 3x speedups, top-1 accuracy

- ❑ over 2%?
- ❑ 1-2%? ?
- ❑ 0.5-1% ?
- ❑ 0.1-0.5%?

Q1: How much progress you expect in the past 6 years for filter/channel pruning?

ResNet50 on ImageNet, 2x - 3x speedups, top-1 accuracy

- ❑ over 2%?
- ❑ 1-2%? ?
- ❑ 0.5-1% ?
- ❑ **0.1-0.5%!**

Table 1. Top-1 accuracy (%) benchmark of filter pruning with **ResNet50** [20] on **ImageNet** [6]. Simply by using a better fine-tuning LR schedule, we manage to revive a *5-year-ago* baseline filter pruning method, L_1*-norm pruning* [32], making it *match or beat* many filter pruning papers published in recent top-tier venues. Note, we achieve this simply by using the common step-decay LR schedule, 90-epoch finetuning, and standard data augmentation, *no* any advanced training recipe (like cosine annealing LR) used. This papers study the reasons and lessons behind this pretty confounding benchmark situation in filter pruning.

Method	Pruned acc. (%)	Speedup
SFP [22] IJCAI'18	74.61	1.72×
DCP [61] NeurIPS'18	74.95	2.25×
GAL-0.5 [37] CVPR'19	71.95	1.76×
Taylor-FO [42] CVPR'19	74.50	1.82×
CCP-AC [45] ICML'19	75.32	2.18×
ProvableFP [35] ICLR'20	75.21	1.43×
HRank [36] CVPR'20	74.98	1.78×
GReg-1 [56] ICLR'21	75.16	2.31×
GReg-2 [56] ICLR'21	75.36	2.31×
CC [34] CVPR'21	**75.59**	2.12×
L_1-norm [32] ICLR'17 **(our reimpl.)**	75.24	**2.31**×
GAL-1 [37] CVPR'19	69.88	2.59×
Factorized [33] CVPR'19	74.55	2.33×
LFPC [21] CVPR'20	74.46	2.55×
GReg-1 [56] ICLR'21	74.85	2.56×
GReg-2 [56] ICLR'21	**74.93**	2.56×
CC [34] CVPR'21	74.54	**2.68**×
L_1-norm [32] ICLR'17 **(our reimpl.)**	74.77	2.56×

At **2 ~ 3x** speedup: The best method only reports **0.16% ~ 0.35%** top-1 accuracy advantage vs. *L1-norm pruning* [Li et al., ICLR, 2017], which is typically considered a very basic filter pruning method.

No particular tricks used. If any, a larger retraining LR (0.01 *vs.* 0.001) and 90 epochs step-decay LR schedule.

The "performance-boosting" effect of a larger retraining LR [1,2]

A larger retraining LR does not really "improve" the performance. What really happens is, pruning damages trainability, then a larger LR accelerates the optimization process in retraining, making the higher performance observed earlier.

Why preserving trainability matters?

1. More faithful pruning benchmarks (less sensitive to hyper-params, e.g., retraining LR).
2. Possibly better pruning performance.

[1] Duong H Le and Binh-Son Hua. Network pruning that matters: A case study on retraining variants. In ICLR, 2021

[2] Comparing rewinding and fine-tuning in neural network pruning. In ICLR, 2020

- Neural network pruning
- Trainability preserving

--- Okay, we are back. Now talk about how to preserve trainability: TPP ---

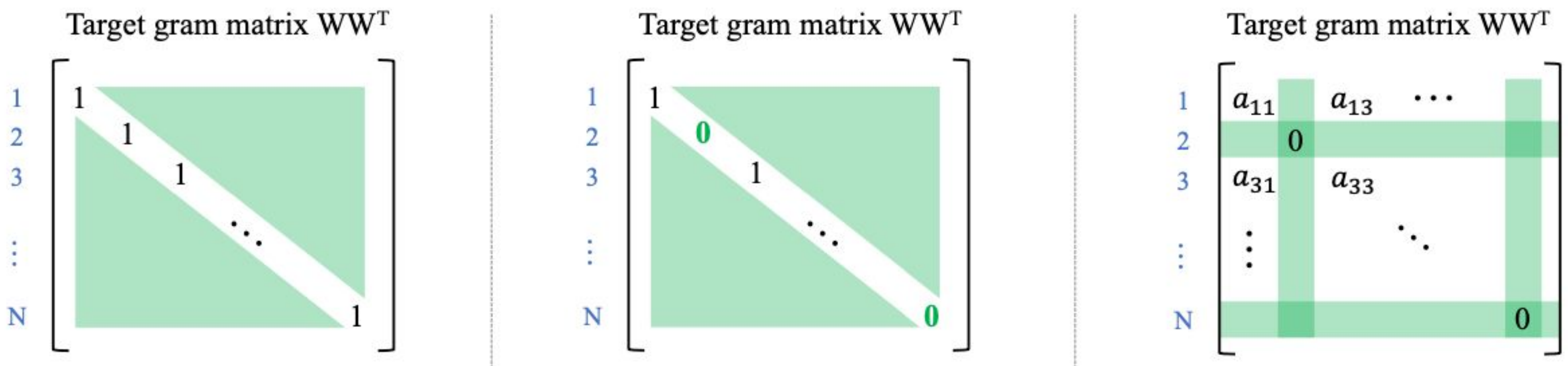
Methodology – Part 1: Regularize the Weight Gram Matrix

- **Trainability vs. orthogonality (norm-preserving)**
  - KernOrth: kernel orthogonality [Xie et al., CVPR, 2017]
  - OrthConv: orthogonal convolution [Wang et al., CVPR, 2020]

$$\begin{aligned} &\mathbf{y} = W\mathbf{x}, \\ &||\mathbf{y}|| = \sqrt{\mathbf{y}^\top \mathbf{y}} = \sqrt{\mathbf{x}^\top W^\top W \mathbf{x}} = ||\mathbf{x}||, \ \ \textit{iff.}\ W^\top W = I, \end{aligned} \tag{1}$$

$$\begin{aligned} &KK^\top = I \Rightarrow \mathcal{L}_{orth} = KK^\top - I, \ \ \triangleleft \ \textbf{kernel orthogonality} \\ &\mathcal{K}\mathcal{K}^\top = I \Rightarrow \mathcal{L}_{orth} = \mathcal{K}\mathcal{K}^\top - I. \ \ \triangleleft \ \textbf{orthogonal convolution} \end{aligned} \tag{2}$$

- Naturally, given the deep connection between trainability and orthogonality, use existing orthogonality methods to help here?
  - KernOrth + L1, L1 + KernOrth
  - OrthConv + L1, L1 + OrthConv



(a) Kernel orthogonality

(b) Kernel orthogonality for pruning

(c) De-correlate pruned from kept

Green = 0
#2, #N filters are pruned

Only penalize the weight gram matrix entries of <u>pruned</u> filters

"we propose to penalize the gram matrix of convolutional filters to <u>decorrelate</u> the pruned filters from the retained filters."

$$\mathcal{L}_1 = \sum_{l=1}^{L} ||W_l W_l^\top \odot (\mathbf{1} - \mathbf{m}\mathbf{m}^\top)||_F^2, \ \mathbf{m}_j = 0 \text{ if } j \in S_l, \ \text{else } 1, \qquad (3)$$

Methodology – Part 2: Regularize BN Parameters

$$f = \gamma \frac{W * X - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta,$$

$$\mathcal{L}_2 = \sum_{l=1}^{L} \sum_{j \in S_l} \gamma_j^2 + \beta_j^2.$$

Only penalize the BN params of <u>pruned</u> filters

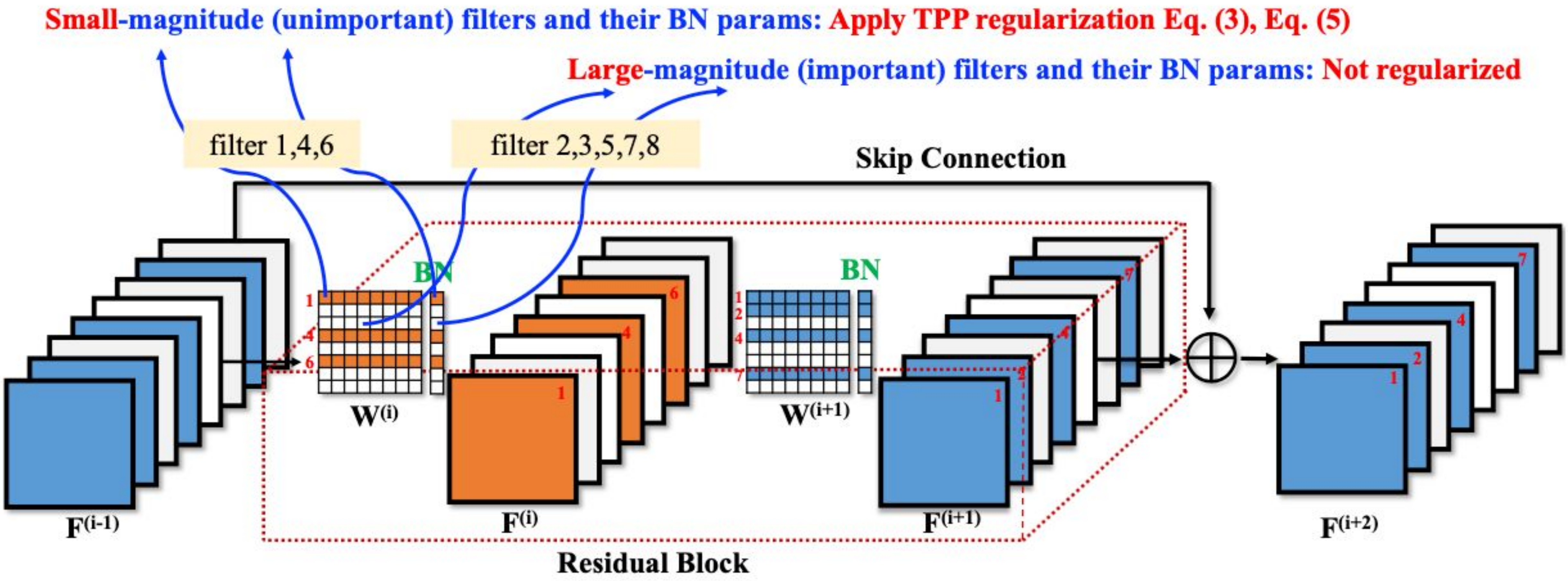
Differences from [1, 2] -- regularize BN scales:

- [1, 2] regularize BN to learn unimportant filters to prune vs. ours: unimportant filters are decided by L1
- [1, 2] only regularize gamma *vs.* ours: regularize both (gamma and **beta**)
  - beta holds a critical role to performance (before fine-tuning)

⇒ We hereby convey an opinion that normalization layers (BN, LN, GN, etc. -- any learnable params) should be considered along with CONV/FC weights, so as to preserve trainability for the whole network.

[1] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learn- ing efficient convolutional networks through network slimming. In ICCV, 2017

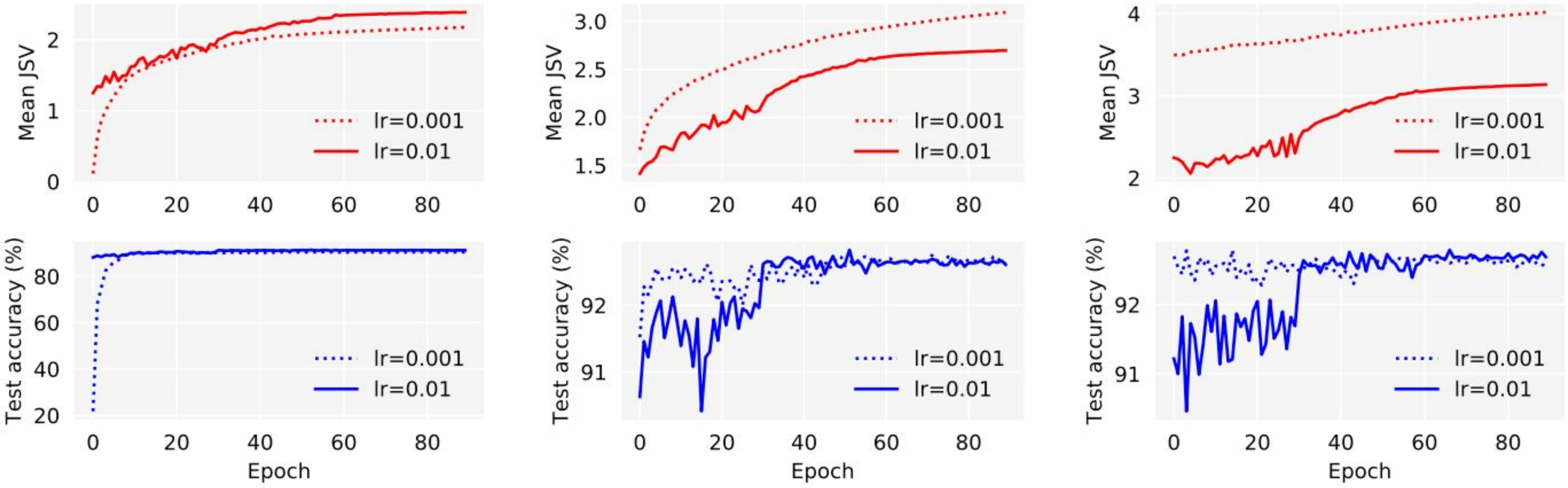
[2] Jianbo Ye, Xin Lu, Zhe Lin, and James Z Wang. Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. In ICLR, 2018.



Overview of our TPP method

Experimental Results

- On small datasets, can TPP preserve trainability better than others?
- On larger dataset (ImageNet-1K), how TPP is compared to other filter pruning methods?



91.36 / 90.54 / 0.0040
(a) L_1

92.79 / 92.77 / 1.0000
(b) L_1+OrthP

92.82 / 92.77 / 3.4875
(c) TPP (ours)

Pruning on MNIST. Below each plot are, in order, the best accuracy of LR 1e-2, the best accuracy of LR 1e-3, and the mean JSV right after pruning (i.e., without retraining)

Table 1: Test accuracy (%) comparison among different isometry maintenance or recovery methods on ResNet56 on CIFAR10. *Scratch* stands for training from scratch. *KernOrth* means Kernel Orthogonalization (Xie et al., 2017); *OrthConv* means Convolutional Orthogonalization (Wang et al., 2020). Two retraining LR schedules are evaluated here: initial LR `1e-2` *vs.* `1e-3`. *Acc. diff.* refers to the accuracy gap of LR `1e-3` against LR `1e-2`.

ResNet56 on CIFAR10: Unpruned acc. 93.78%, Params: 0.85M, FLOPs: 0.25G					
Layerwise PR	0.3	0.5	0.7	0.9	0.95
Sparsity/Speedup	31.14%/1.45×	49.82%/1.99×	70.57%/3.59×	90.39%/11.41×	95.19%/19.31×
			`Initial retraining LR 1e-2`		
Scratch	93.16 (0.16)	92.78 (0.23)	92.11 (0.12)	<u>88.36</u> (0.20)	<u>84.60</u> (0.14)
L_1 (Li et al., 2017)	<u>93.79</u> (0.06)	**93.51** (0.07)	92.26 (0.17)	86.75 (0.31)	83.03 (0.07)
L_1 + OrthP (Wang et al., 2021a)	93.69 (0.02)	93.36 (0.19)	91.96 (0.06)	86.01 (0.34)	82.62 (0.05)
L_1 + KernOrth (Xie et al., 2017)	93.49 (0.04)	93.30 (0.19)	91.71 (0.14)	84.78 (0.34)	80.87 (0.47)
L_1 + OrthConv (Wang et al., 2020)	92.54 (0.09)	92.41 (0.07)	91.02 (0.16)	84.52 (0.27)	80.23 (1.19)
KernOrth (Xie et al., 2017) + L_1	93.49 (0.07)	92.82 (0.10)	90.54 (0.25)	85.47 (0.20)	79.48 (0.81)
OrthConv (Wang et al., 2020) + L_1	93.63 (0.17)	93.28 (0.20)	<u>92.27</u> (0.13)	86.70 (0.07)	83.21 (0.61)
TPP (ours)	**93.81** (0.11)	<u>93.46</u> (0.06)	**92.35** (0.12)	**89.63** (0.10)	**85.86** (0.08)
			`Initial retraining LR 1e-3`		
L_1 (Li et al., 2017)	93.43 (0.06)	93.12 (0.10)	91.77 (0.11)	87.57 (0.09)	83.10 (0.12)
TPP (ours)	**93.54** (0.08)	**93.32** (0.11)	**92.00** (0.08)	**89.09** (0.10)	**85.47** (0.22)
Acc. diff. (L_1)	-0.38	-0.40	-0.50	+0.82	+0.07
Acc. diff. (TPP)	-0.27	-0.14	-0.35	-0.54	-0.39

Table 2: Comparison on ImageNet-1K validation set. *Advanced training recipe (such as cosine LR schedule) is used; we single them out for fair comparison.

Method	Model	Unpruned top-1 (%)	Pruned top-1 (%)	Top-1 drop (%)	Speedup
L_1 (pruned-B) (Li et al., 2017)	ResNet34	73.23	72.17	1.06	**1.32**×
L_1 (pruned-B, reimpl.) (Wang et al., 2023)		73.31	73.67	-0.36	**1.32**×
Taylor-FO (Molchanov et al., 2019)		73.31	72.83	0.48	1.29×
GReg-2 (Wang et al., 2021b)		73.31	73.61	-0.30	**1.32**×
TPP (ours)		73.31	**73.77**	**-0.46**	**1.32**×
ProvableFP (Liebenwein et al., 2020)	ResNet50	76.13	75.21	0.92	1.43×
MetaPruning (Liu et al., 2019a)		76.6	76.2	0.4	1.37×
GReg-1 (Wang et al., 2021b)		76.13	76.27	-0.14	**1.49**×
TPP (ours)		76.13	**76.44**	**-0.31**	**1.49**×
IncReg (Wang et al., 2019)	ResNet50	75.60	72.47	3.13	2.00×
SFP (He et al., 2018)		76.15	74.61	1.54	1.72×
HRank (Lin et al., 2020)		76.15	74.98	1.17	1.78×
Taylor-FO (Molchanov et al., 2019)		76.18	74.50	1.68	1.82×
Factorized (Li et al., 2019)		76.15	74.55	1.60	**2.33**×
DCP (Zhuang et al., 2018)		76.01	74.95	1.06	2.25×
CCP-AC (Peng et al., 2019)		76.15	75.32	0.83	2.18×
GReg-2 (Wang et al., 2021b)		76.13	75.36	0.77	2.31×
CC (Li et al., 2021)		76.15	75.59	0.56	2.12×
MetaPruning (Liu et al., 2019a)		76.6	75.4	1.2	2.00×
TPP (ours)		76.13	**75.60**	**0.53**	2.31×
LFPC (He et al., 2020)	ResNet50	76.15	74.46	1.69	2.55×
GReg-2 (Wang et al., 2021b)		76.13	74.93	1.20	2.56×
CC (Li et al., 2021)		76.15	74.54	1.61	**2.68**×
TPP (ours)		76.13	**75.12**	**1.01**	2.56×
IncReg (Wang et al., 2019)	ResNet50	75.60	71.07	4.53	3.00×
Taylor-FO (Molchanov et al., 2019)		76.18	71.69	4.49	3.05×
GReg-2 (Wang et al., 2021b)		76.13	73.90	2.23	**3.06**×
TPP (ours)		76.13	**74.51**	**1.62**	**3.06**×

Method	Network	Top-1 (%)	FLOPs (G)
CHEX* (Hou et al., 2022)	ResNet50	77.4	2
CHEX* (Hou et al., 2022)		76.0	1
TPP* (ours)		**77.75**	2
TPP* (ours)		**76.52**	1

At 2x ~ 3x speedup, **0.1 ~ 0.6%** top1 acc improvement than the last SOTA

Limitations

- TPP is definitely not the final answer (not even close…)
- Trainability preserving pruning -- one-shot pruning (pruning as initialization)
  - (1) No data available (trainability describe a property of the network per se)
  - (2) No iterative optimization
- Weaker conditions:
  - (1) Pseudo data (from pretrained model, data-free knowledge distillation?) + iterative optimization
  - (2) One-shot + data available -- E.g., *SparseGPT: Massive Language Models Can Be Accurately Pruned in One-Shot* (https://arxiv.org/abs/2301.00774)

Code: https://github.com/MingSun-Tse/TPP
Google “TPP, Huan Wang, Github”

Conclusion

1. We present **TPP** (trainability preserving pruning), a new filter pruning method.
2. **Two components**: Regularizing weight gram matrix + regularizing BN (only apply penalty to pruned entries)
3. TPP is the ***first*** trainability preserving method scalable to ImageNet-1K.
4. Limitations: One-shot pruning method that can preserve trainability on ImageNet is yet to see.

Thanks for your attention!