Why is the State of Neural Network Pruning so Confusing?

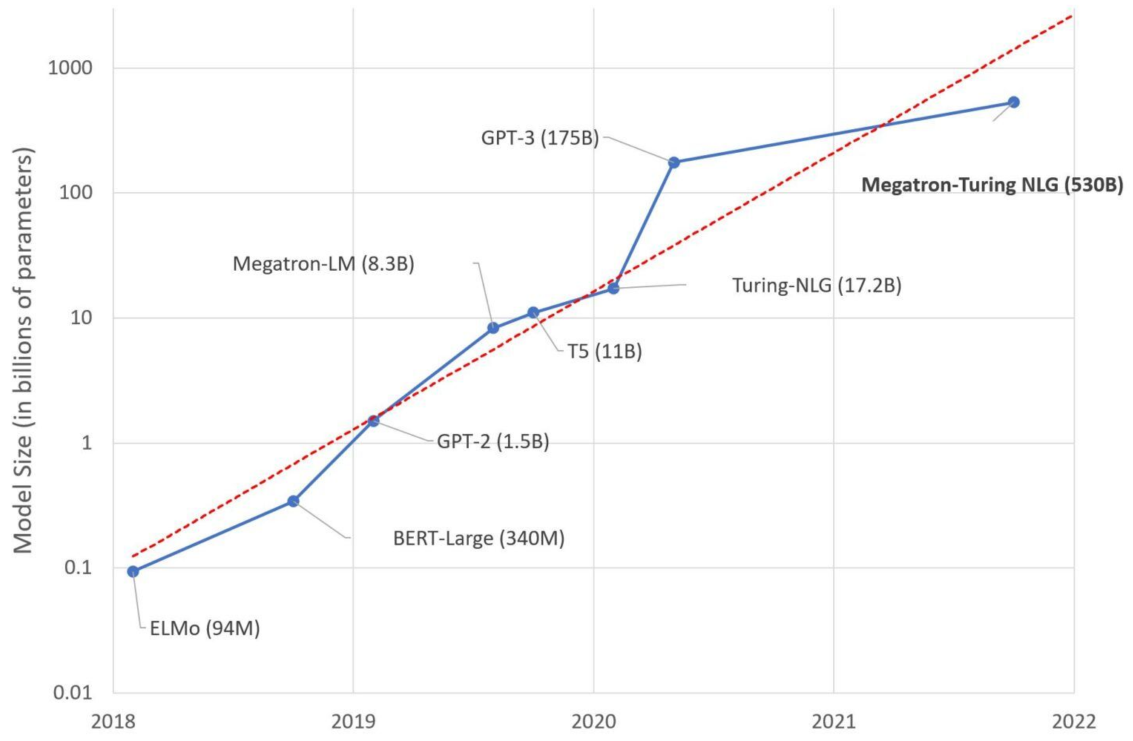
On the Fairness, Comparison Setup, and Trainability in Network Pruning

Huan Wang   Can Qin   Yue Bai   Yun Fu

Northeastern University, Boston, MA

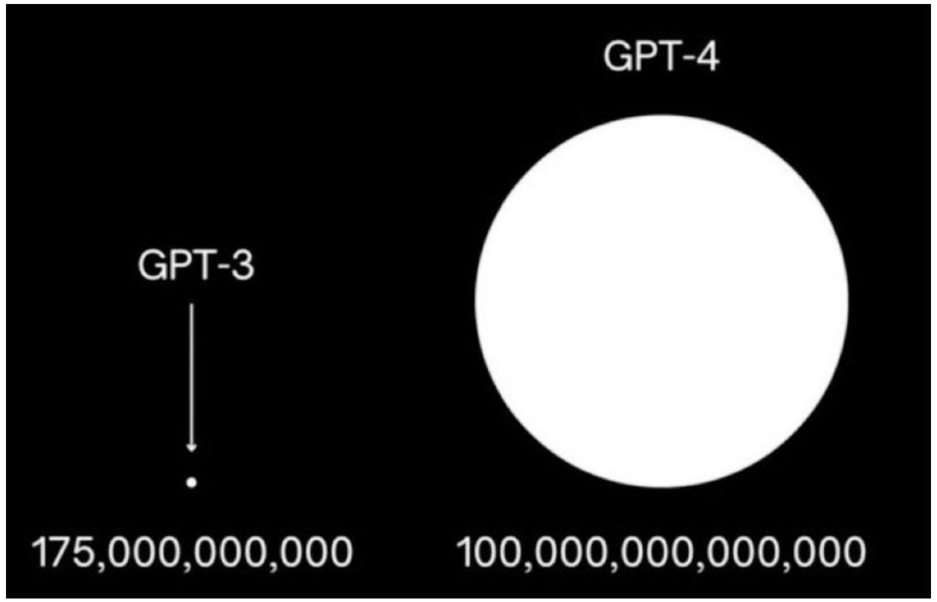
Mar 19, 2023

We are in the era of *large* models
(and they are growing quickly)



Trend of sizes of state-of-the-art NLP models over time

src: https://www.microsoft.com/en-us/research/blog/using-deepspeed-and-megatron-to-train-megatron-turing-nlg-530b-the-worlds-largest-and-most-powerful-generative-language-model/



A widely-spread diagram (turns out to be misinformation), indicating the giant size of GPT-4 to come

Do we really need so many parameters? No!

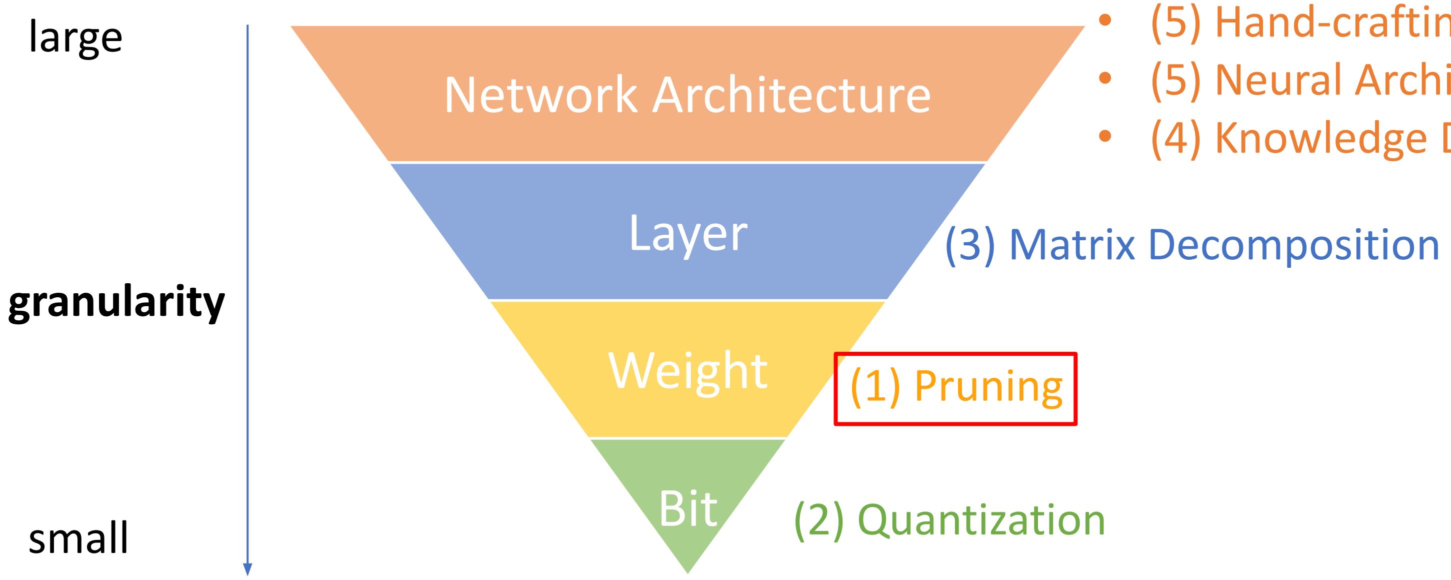
Model Compression or **Efficient Deep Learning**

Make the model smaller/faster while maintaining the performance

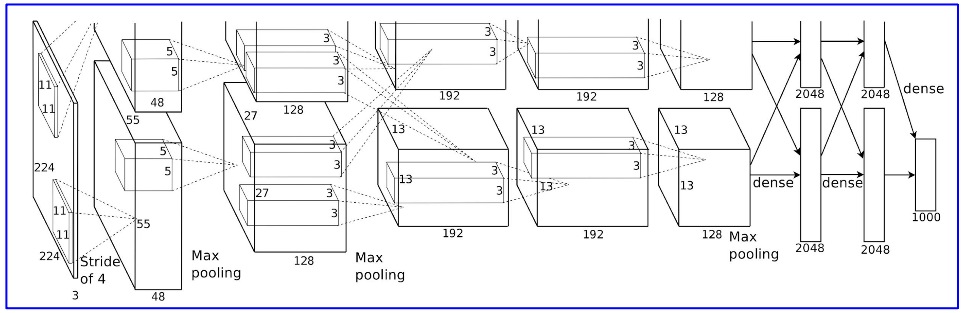
Categories of Model Compression

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) (6) Arithmetic complexity reduction

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



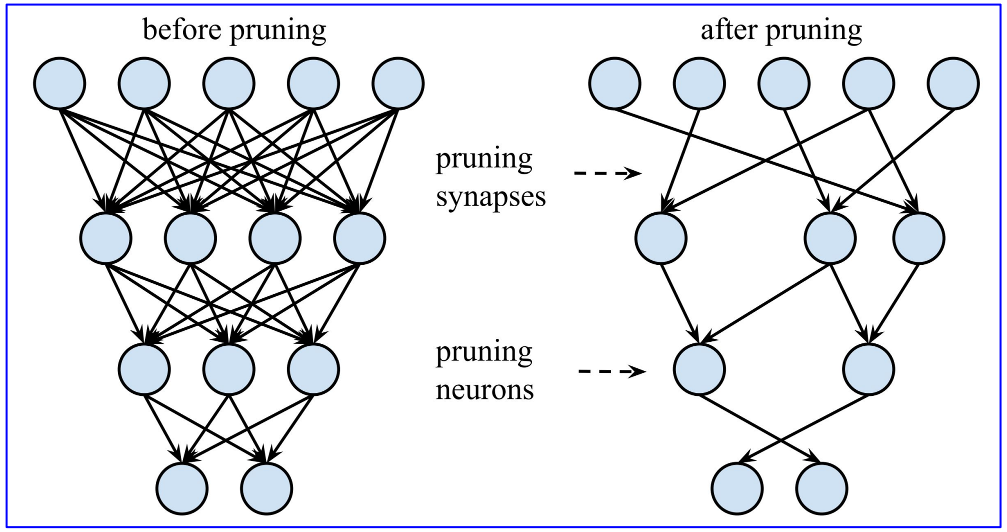
Hierarchical Redundancy of DNNs



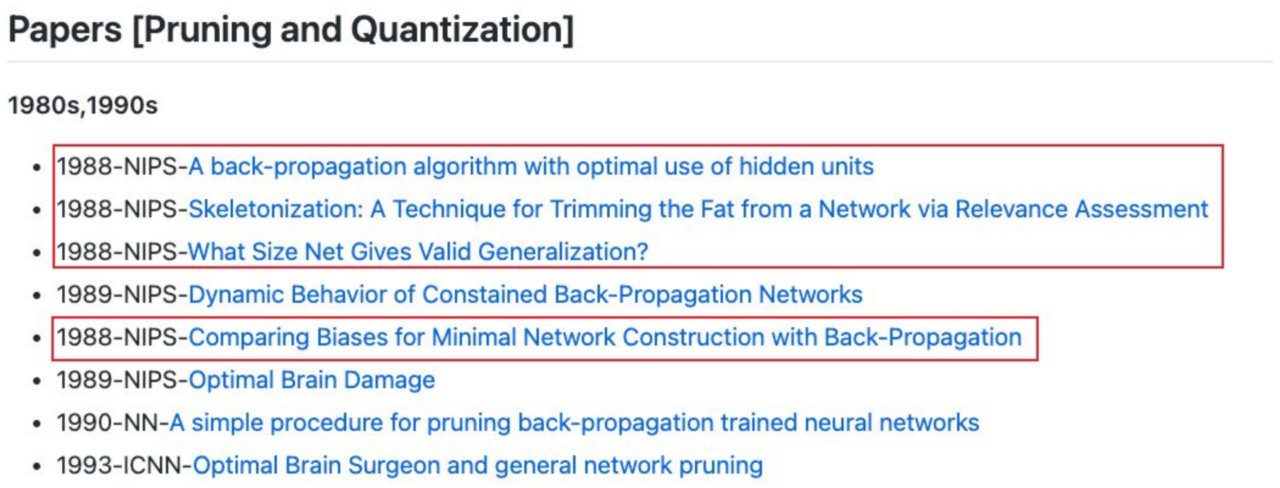
[AlexNet, 2012, NIPS]

[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).

History of Network Pruning



[Han et al., NIPS, 2015]



Papers [Pruning and Quantization]

1980s,1990s

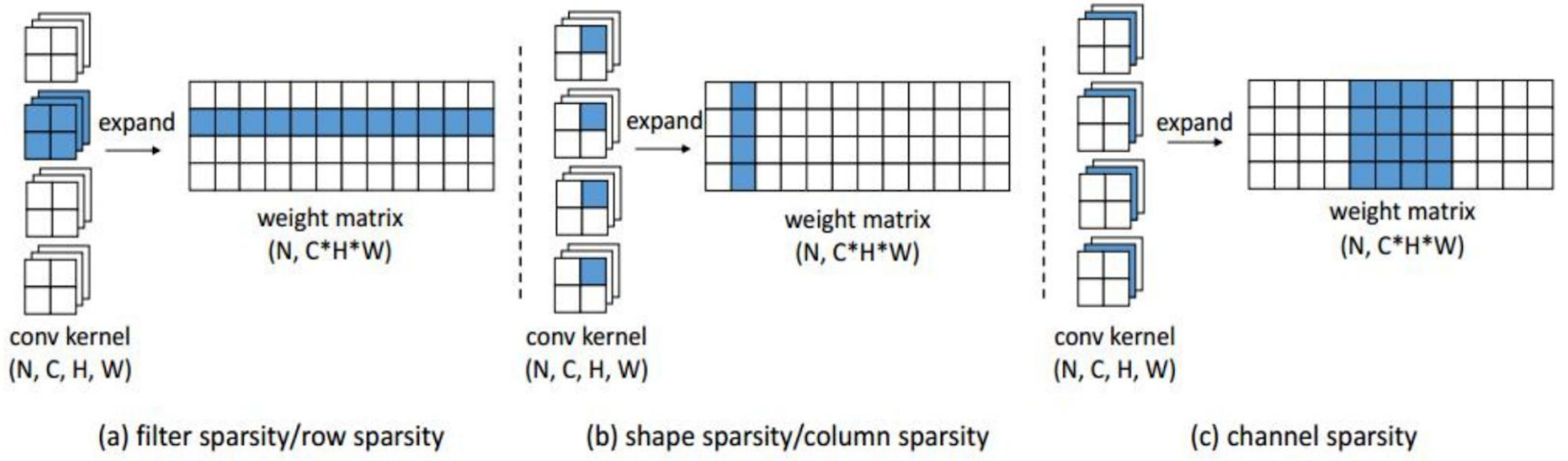
- 1988-NIPS-A back-propagation algorithm with optimal use of hidden units
- 1988-NIPS-Skeletonization: A Technique for Trimming the Fat from a Network via Relevance Assessment
- 1988-NIPS-What Size Net Gives Valid Generalization?
- 1989-NIPS-Dynamic Behavior of Constained Back-Propagation Networks
- 1988-NIPS-Comparing Biases for Minimal Network Construction with Back-Propagation
- 1989-NIPS-Optimal Brain Damage
- 1990-NN-A simple procedure for pruning back-propagation trained neural networks
- 1993-ICNN-Optimal Brain Surgeon and general network pruning

src: https://github.com/MingSun-Tse/Efficient-Deep-Learning

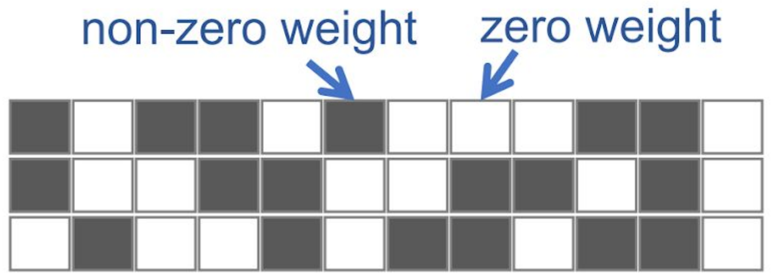
- Pruning is probably the earliest mode compression method among the five.
  - In 1986, BP was popularized for training neural networks [Rumelhart et al., Nature, 1986].
  - **In 1987, 1st NIPS (now NeurIPS) conference.**
  - **In 1988, pruning papers emerged.**
  - In 1989 NIPS, LeCun et al. proposed Optimal Brain Damage for pruning.
  - Past focus: Generalization, not faster. Nowadays: smaller and faster (because generalization problem has been largely resolved via big model + big data + NVIDIA GPUs)

Structured pruning (hardware-friendly)

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



Structured pruning



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

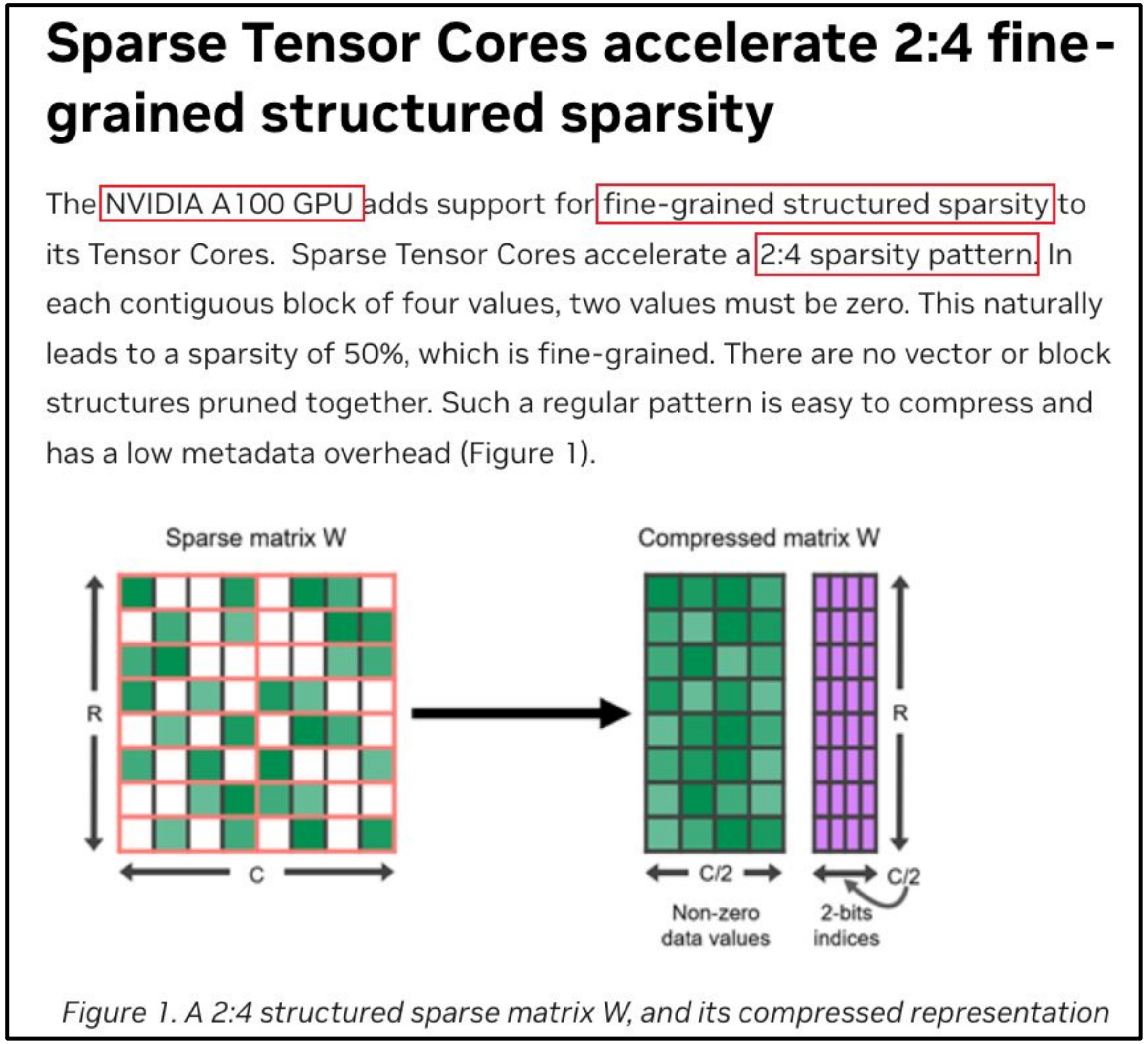
Regularity (Constraint)-Performance: It's all about *trade-off*!

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

What is considered “structured” depends on the hardware condition



Sparse Tensor Cores accelerate 2:4 fine-grained structured sparsity

The NVIDIA A100 GPU adds support for fine-grained structured sparsity to its Tensor Cores. Sparse Tensor Cores accelerate a 2:4 sparsity pattern. In each contiguous block of four values, two values must be zero. This naturally leads to a sparsity of 50%, which is fine-grained. There are no vector or block structures pruned together. Such a regular pattern is easy to compress and has a low metadata overhead (Figure 1).

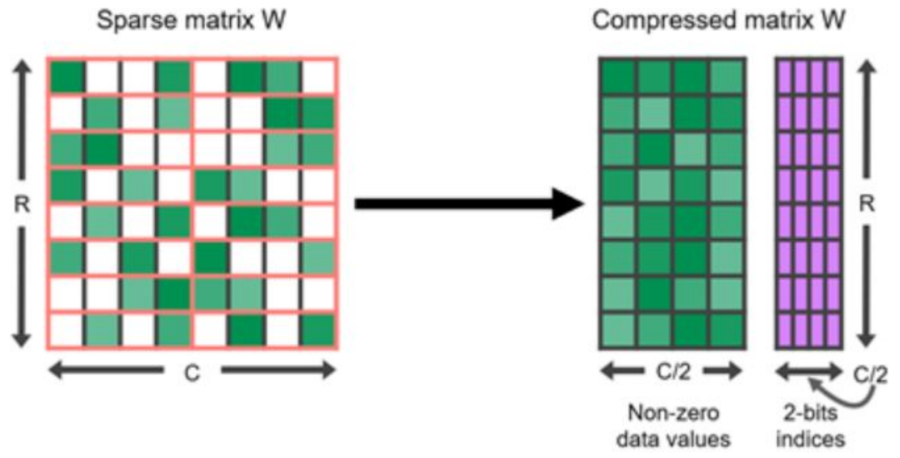
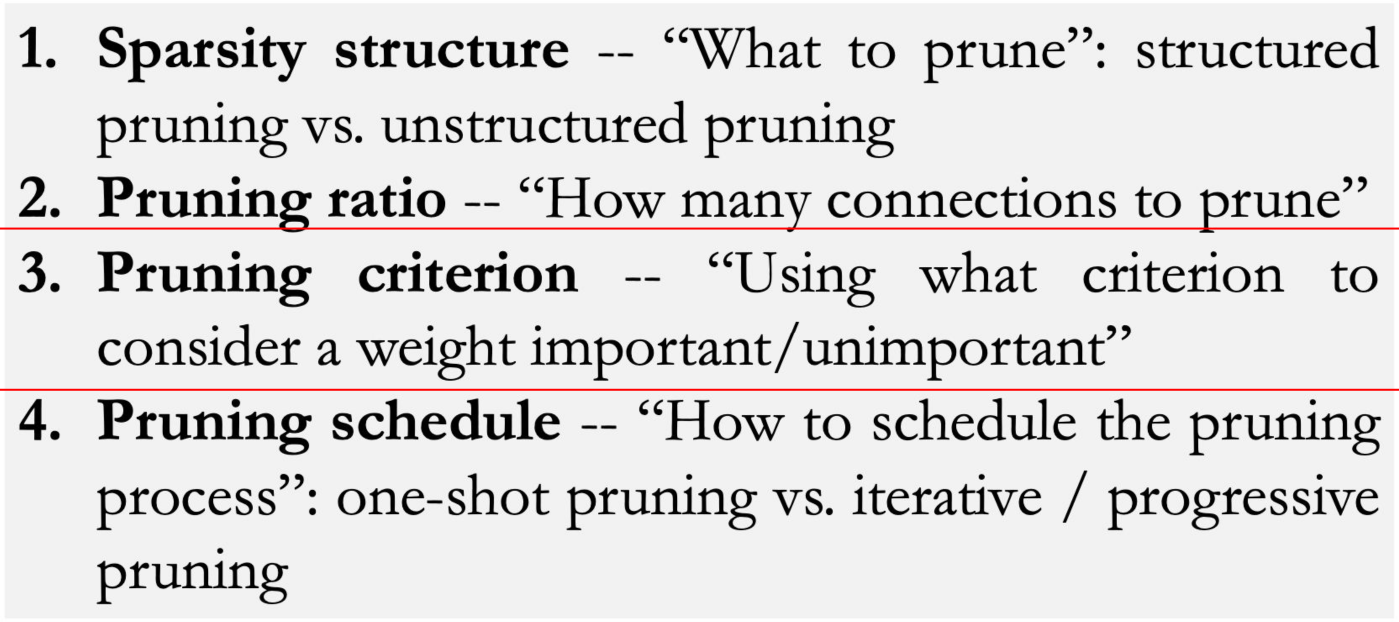


Figure 1. A 2:4 structured sparse matrix W, and its compressed representation

Free lunch -- 2:4 sparsity on A100 GPUs.

src: https://developer.nvidia.com/blog/accelerating-inference-with-sparsity-using-ampere-and-tensorrt/

Four Key Questions in Network Pruning



1. **Sparsity structure** -- "What to prune": structured pruning vs. unstructured pruning
2. **Pruning ratio** -- "How many connections to prune"
3. **Pruning criterion** -- "Using what criterion to consider a weight important/unimportant"
4. **Pruning schedule** -- "How to schedule the pruning process": one-shot pruning vs. iterative / progressive pruning

Arguably the most important questions

src: [Wang et al., IJCAI, 2022]

[Wang et al., IJCAI, 2022] Wang, Huan, et al. "Recent Advances on Neural Network Pruning at Initialization." in IJCAI, 2022

Pruning is typically conducted on pretrained model, while recently some emerging pruning fashions are not. They prune randomly initialized models -- **Pruning at Initialization.**

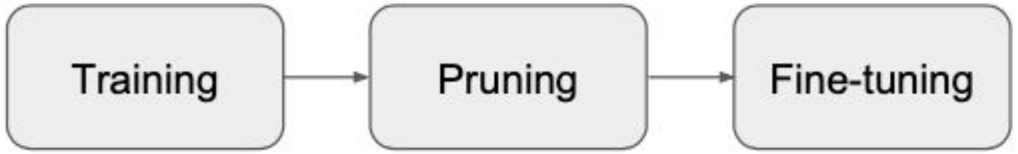
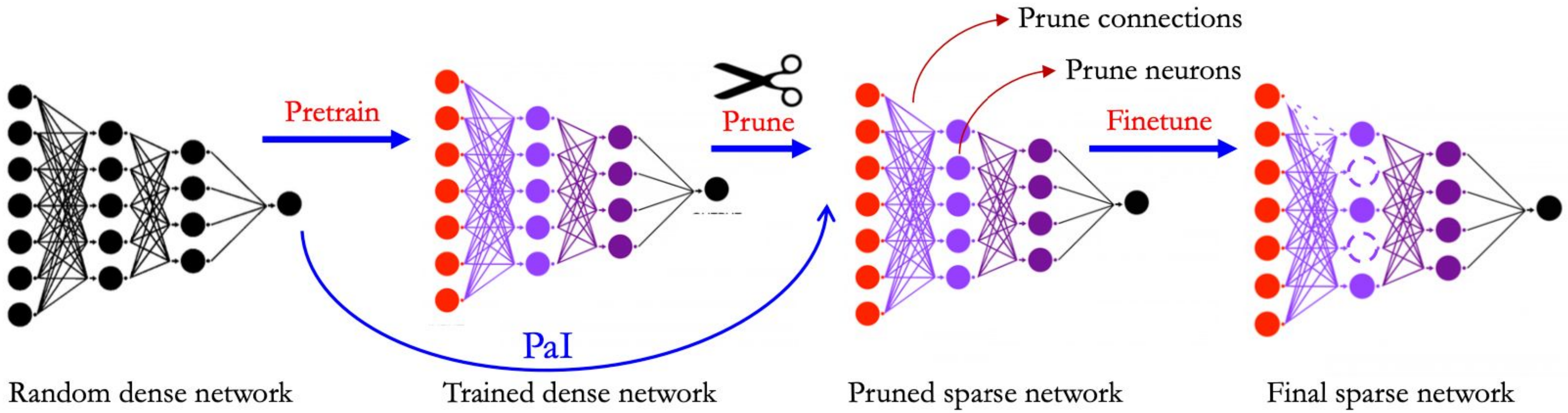


Figure 1: A typical three-stage network pruning pipeline.

[Liu et al., ICLR, 2019]

Background: What is Pruning at Initialization (PaI)?



The "conventional" 3-step pruning pipeline: **Pruning after training (PaT)**
Pruning at initialization (PaI) **saves the first step**, while maintaining comparable performance to PaT.

History Sketch: Debut of PaI (2 papers)

- ❑ Lottery Ticket Hypothesis (LTH) [Frankle and Carbin, ICLR, 2019] (**Best paper award**)
- ❑ Single-shot Network Pruning (SNIP) [Lee et al., ICLR, 2019]

Get mask via MP (magnitude pruning)

Post-selected

- LTH: F0 (Rand, Dense) ➔ F1 (Trained, Dense) ➔ Mask1; F0 * Mask1 ➔ F2 (Rand, Sparse) ➔ F3 (Trained, Sparse)
- SNIP: F0 * Mask2 ➔ F2 (Rand, Sparse) ➔ F3 (Trained, Sparse)

They both claim: F3 performs comparbaly to F1.

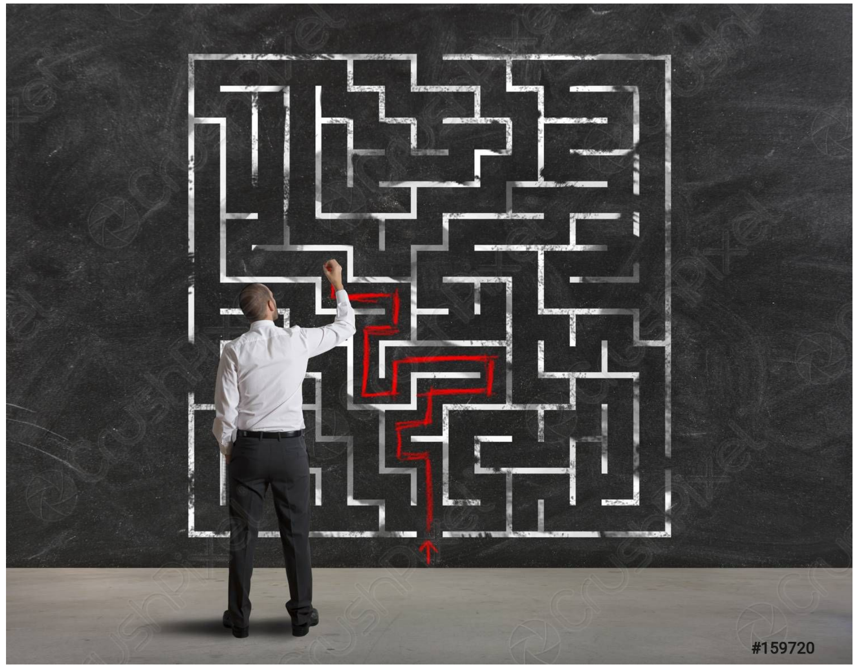
Pre-selected

Non-trivially sparse networks can be trained to full accuracy in isolation.

"PaI = Dense"

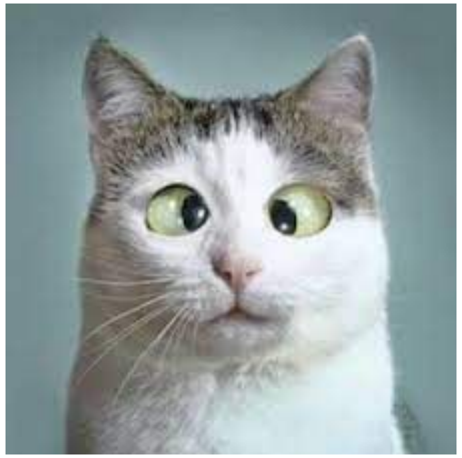
Src: [Wang et al., IJCAI, 2022] "Recent Advances on Neural Network Pruning at Initialization"

Pretty many mysteries in the area of network (filter) pruning. This work is to unveil two of them.



Play a maze game!

Image you know nothing about network pruning and try to find some fantastic pruning algorithm to compress / accelerate your model.



“I know nothing about network pruning”

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%!**

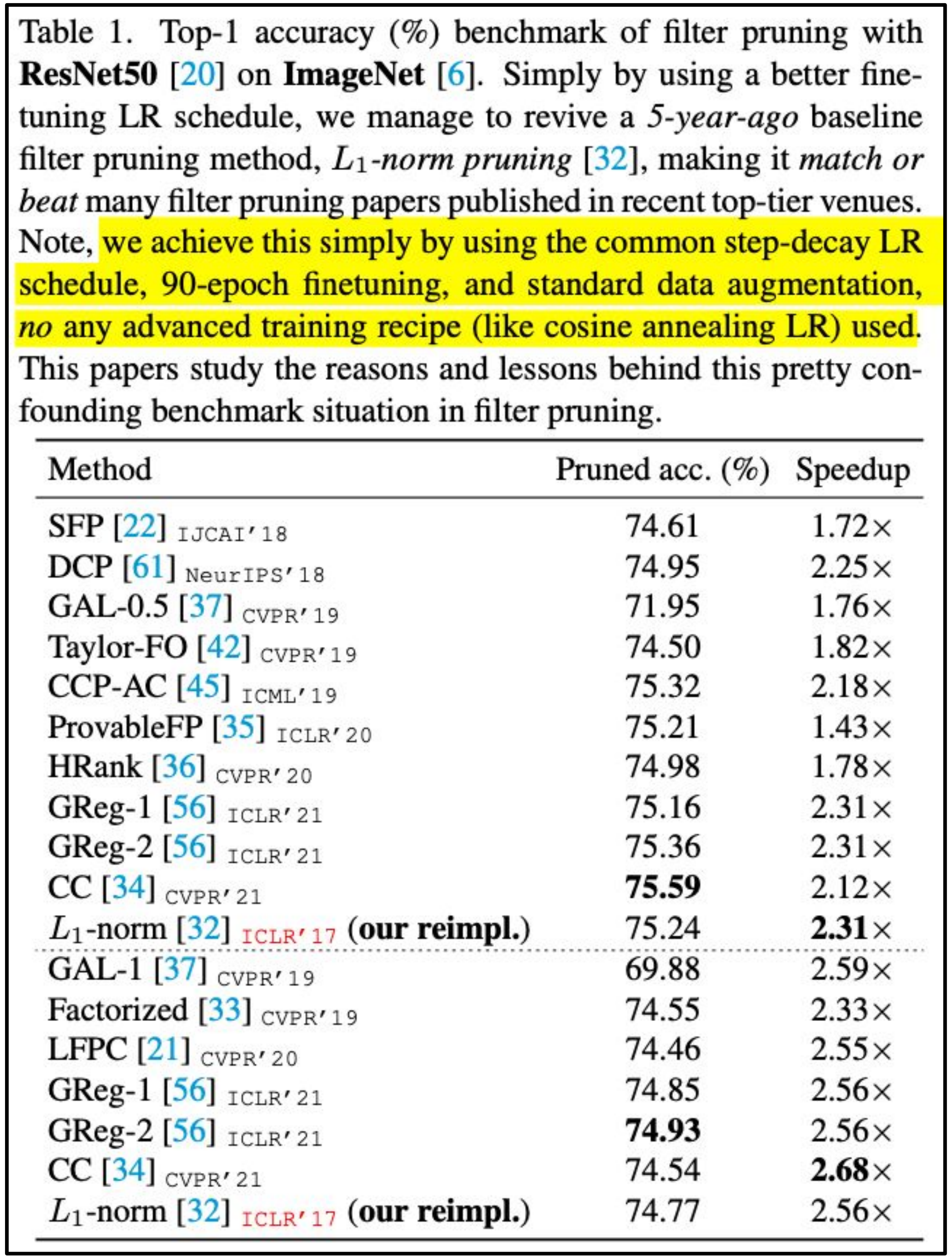
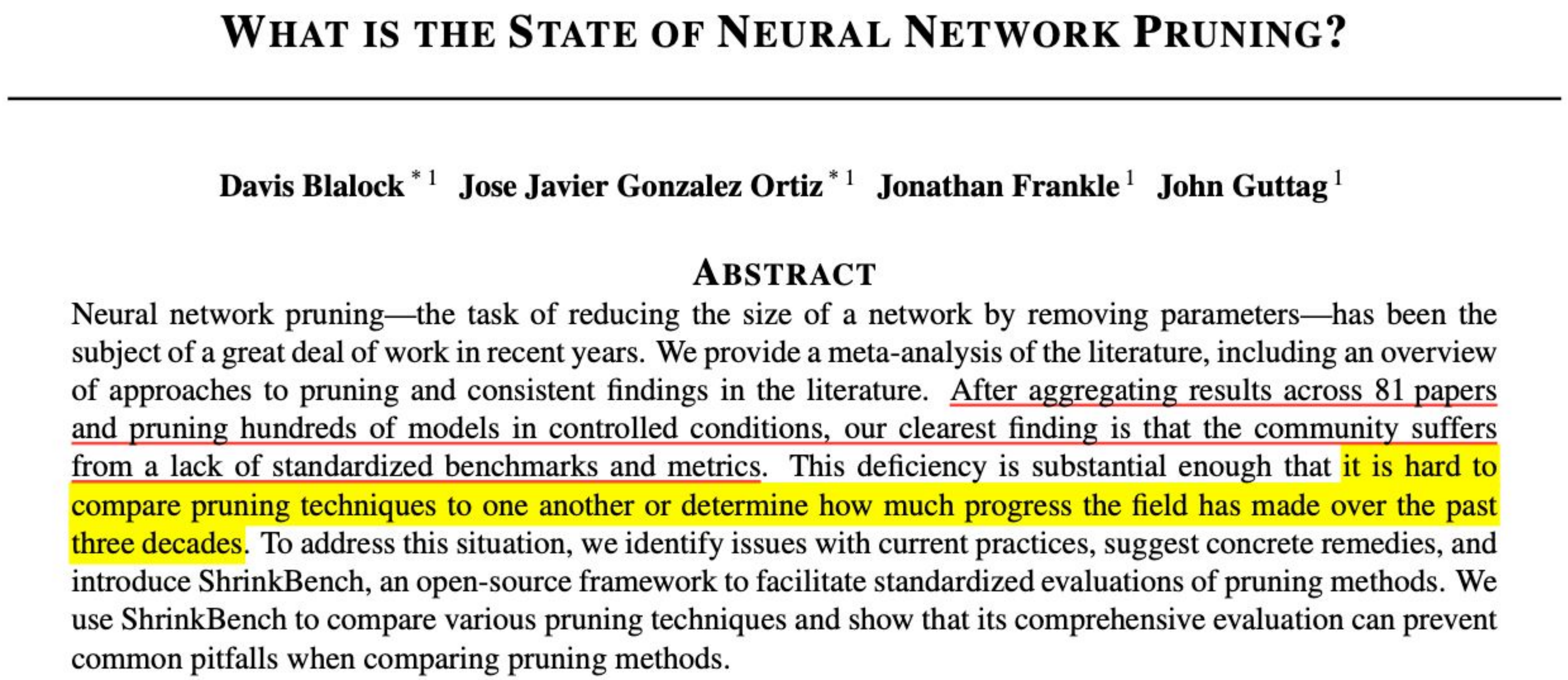


Table 1. Top-1 accuracy (%) benchmark of filter pruning with **ResNet50** [20] on **ImageNet** [6]. Simply by using a better fine-tuning LR schedule, we manage to revive a *5-year-ago* baseline filter pruning method, L_1*-norm pruning* [32], making it *match or beat* many filter pruning papers published 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 fine tuning LR (0.01 vs. 0.001) and 90 epochs step-decay LR schedule.



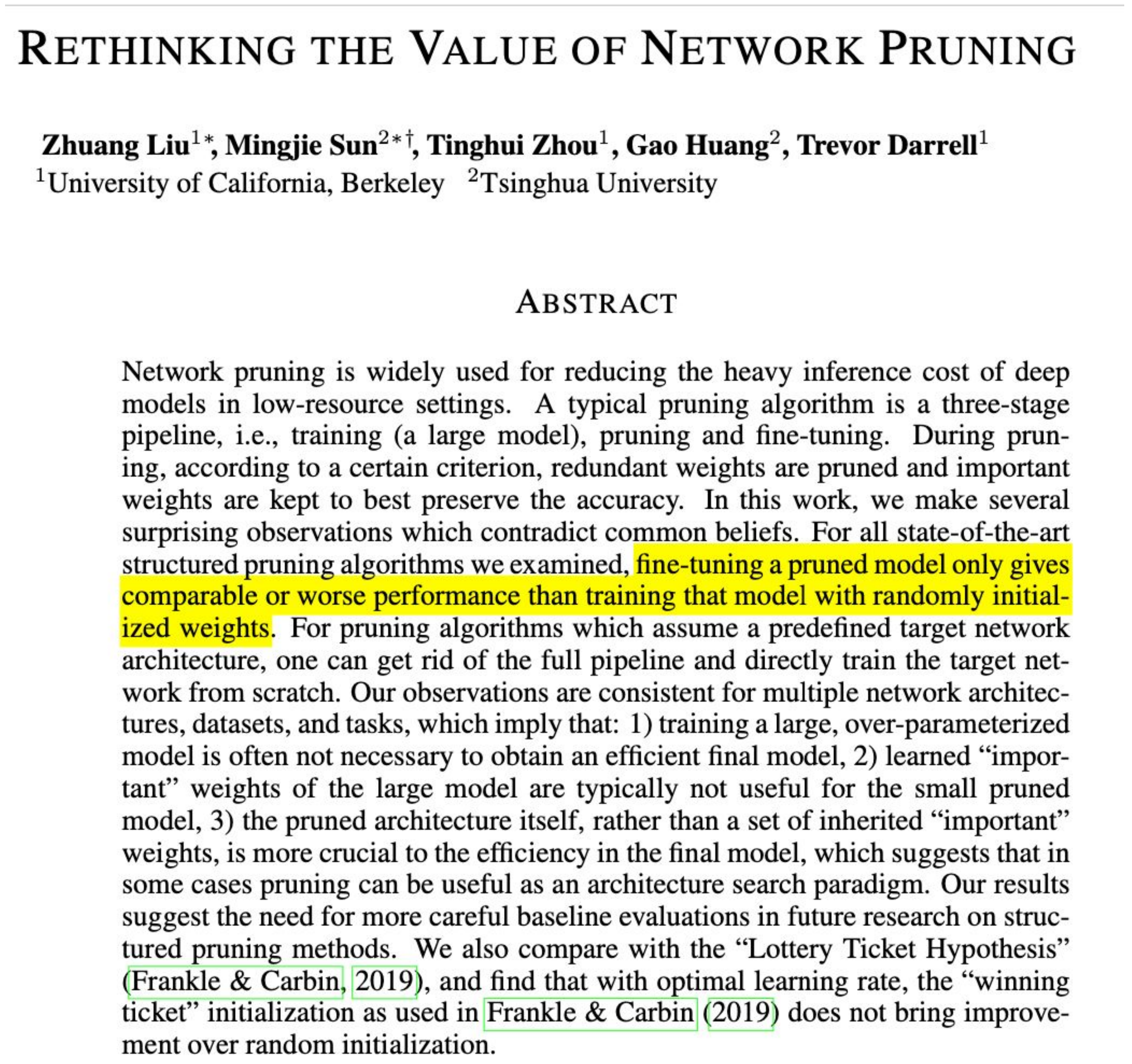
What is the State of Neural Network Pruning?

Davis Blalock [*1] Jose Javier Gonzalez Ortiz [*1] Jonathan Frankle [1] John Guttag [1]

Abstract

Neural network pruning—the task of reducing the size of a network by removing parameters—has been the subject of a great deal of work in recent years. We provide a meta-analysis of the literature, including an overview of approaches to pruning and consistent findings in the literature. After aggregating results across 81 papers and pruning hundreds of models in controlled conditions, our clearest finding is that the community suffers from a lack of standardized benchmarks and metrics. This deficiency is substantial enough that it is hard to compare pruning techniques to one another or determine how much progress the field has made over the past three decades. To address this situation, we identify issues with current practices, suggest concrete remedies, and introduce ShrinkBench, an open-source framework to facilitate standardized evaluations of pruning methods. We use ShrinkBench to compare various pruning techniques and show that its comprehensive evaluation can prevent common pitfalls when comparing pruning methods.

[Blalock et al., MLsys, 2020]



RETHINKING THE VALUE OF NETWORK PRUNING

Zhuang Liu[1]*, **Mingjie Sun**[2]*†, **Tinghui Zhou**[1], **Gao Huang**[2], **Trevor Darrell**[1]
[1]University of California, Berkeley [2]Tsinghua University

ABSTRACT

Network pruning is widely used for reducing the heavy inference cost of deep models in low-resource settings. A typical pruning algorithm is a three-stage pipeline, i.e., training (a large model), pruning and fine-tuning. During pruning, according to a certain criterion, redundant weights are pruned and important weights are kept to best preserve the accuracy. In this work, we make several surprising observations which contradict common beliefs. For all state-of-the-art structured pruning algorithms we examined, fine-tuning a pruned model only gives comparable or worse performance than training that model with randomly initialized weights. For pruning algorithms which assume a predefined target network architecture, one can get rid of the full pipeline and directly train the target network from scratch. Our observations are consistent for multiple network architectures, datasets, and tasks, which imply that: 1) training a large, over-parameterized model is often not necessary to obtain an efficient final model, 2) learned "important" weights of the large model are typically not useful for the small pruned model, 3) the pruned architecture itself, rather than a set of inherited "important" weights, is more crucial to the efficiency in the final model, which suggests that in some cases pruning can be useful as an architecture search paradigm. Our results suggest the need for more careful baseline evaluations in future research on structured pruning methods. We also compare with the "Lottery Ticket Hypothesis" (Frankle & Carbin, 2019), and find that with optimal learning rate, the "winning ticket" initialization as used in Frankle & Carbin (2019) does not bring improvement over random initialization.

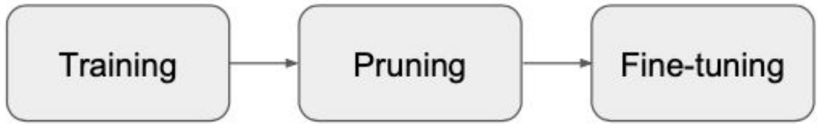


Figure 1: A typical three-stage network pruning pipeline.

M2: No value of the pretraining step in network filter pruning, which radically challenges the past belief.

There is no point doing filter pruning! Just train from scratch!

The best way to do pruning is do nothing.

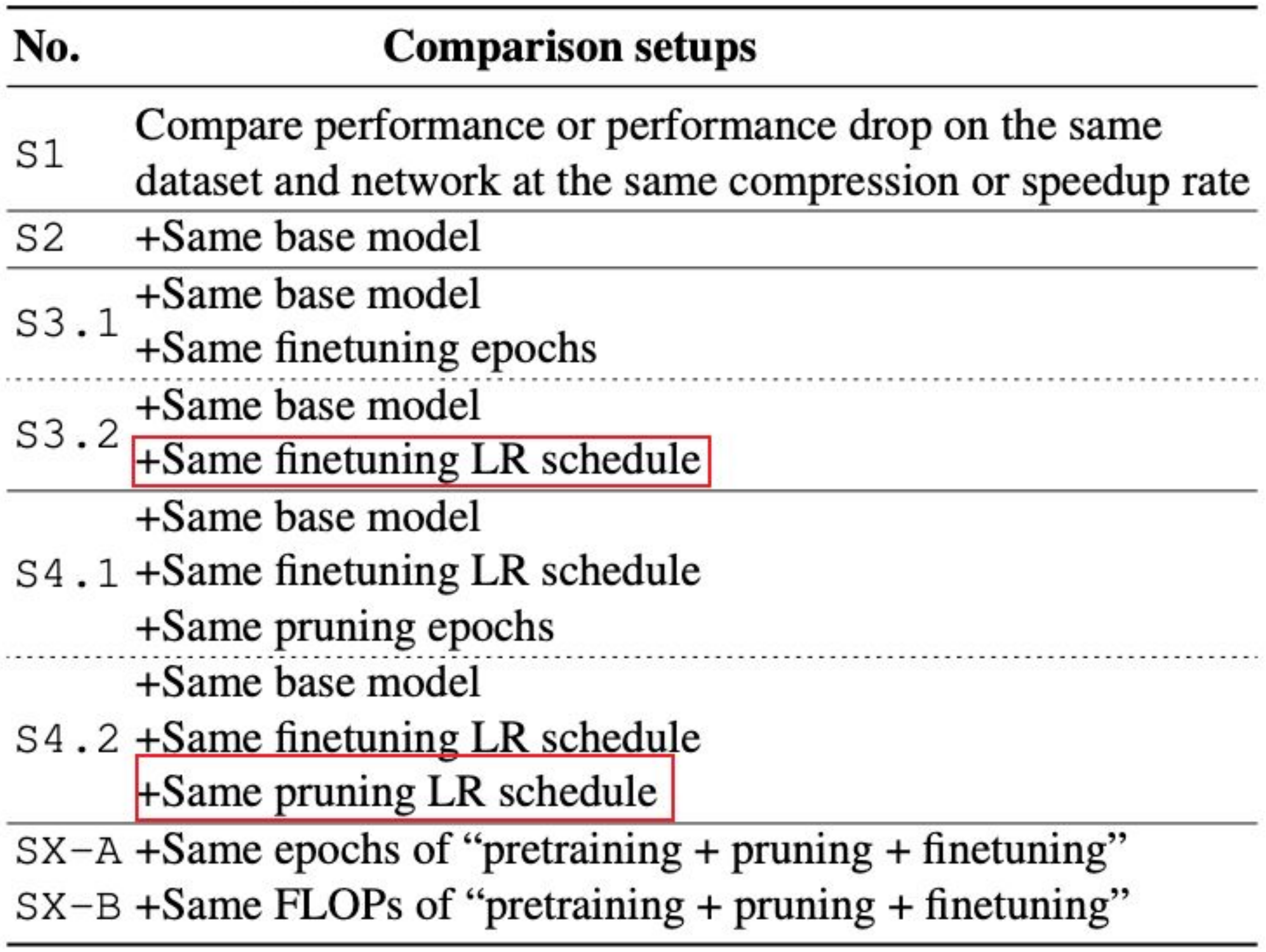
Is that true? Why I still see filter pruning papers popping up since ICLR’19?

“Use your common sense”

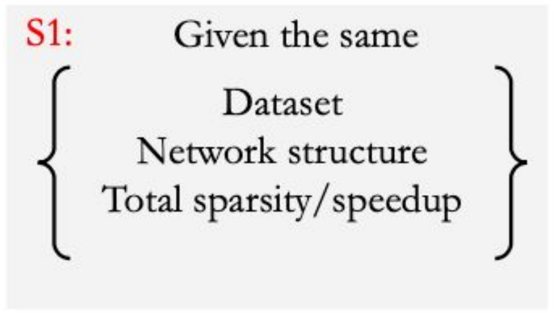
Road Map

- Comparison setups in pruning papers.
- Unveil M2 first (as we’ll show, M2 reduces to M1).
- Then unveil M1. Keyword: Network trainability
- Conclusion and takeaways.

Pruning History 101



No.	Comparison setups
S1	Compare performance or performance drop on the same dataset and network at the same compression or speedup rate
S2	+Same base model
S3.1	+Same base model +Same finetuning epochs
S3.2	+Same base model +Same finetuning LR schedule
S4.1	+Same base model +Same finetuning LR schedule +Same pruning epochs
S4.2	+Same base model +Same finetuning LR schedule +Same pruning LR schedule
SX-A	+Same epochs of “pretraining + pruning + finetuning”
SX-B	+Same FLOPs of “pretraining + pruning + finetuning”



S1: Given the same

$$\left\{ \begin{array}{c} \text{Dataset} \\ \text{Network structure} \\ \text{Total sparsity/speedup} \end{array} \right\}$$

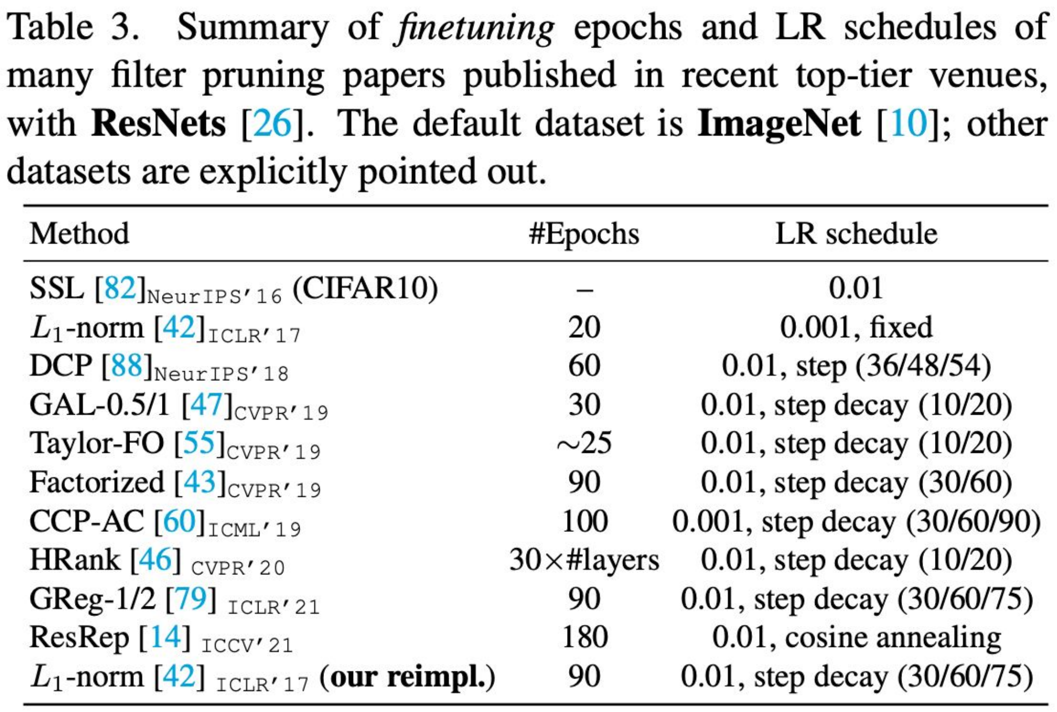
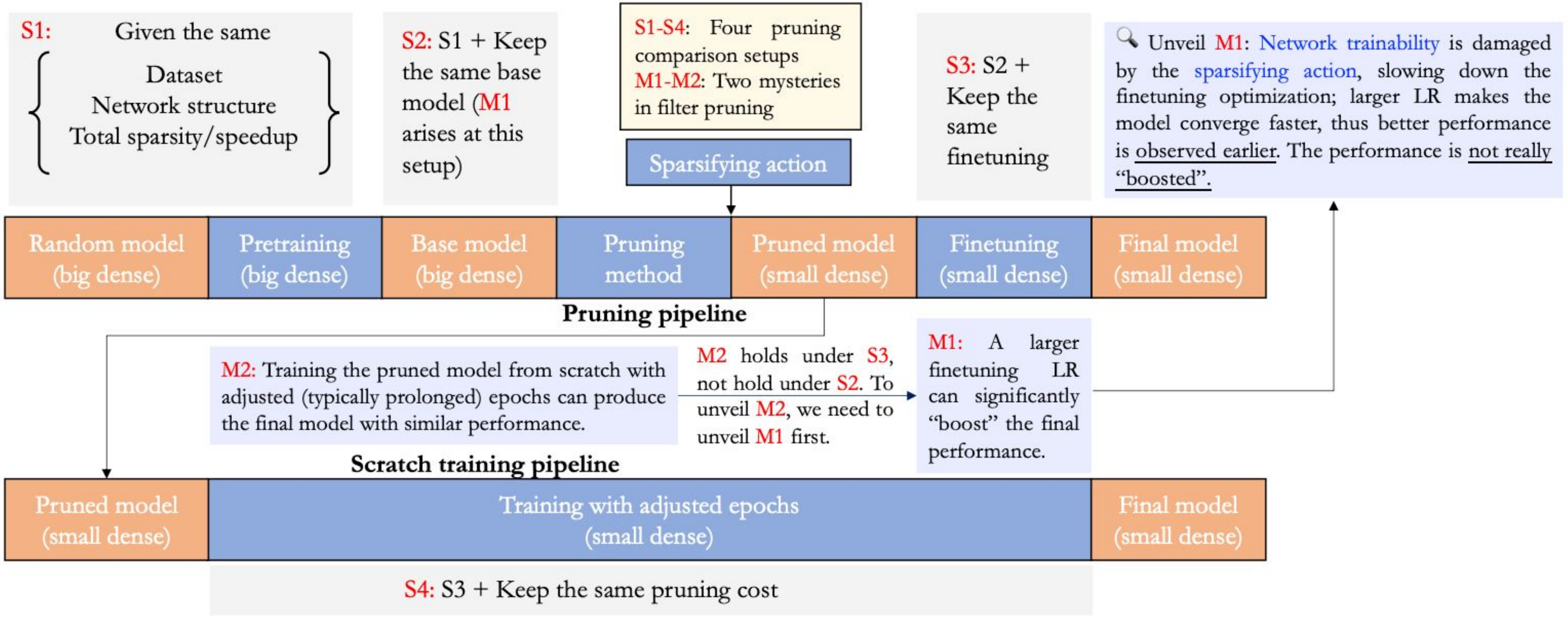


Table 3. Summary of *finetuning* epochs and LR schedules of many filter pruning papers published in recent top-tier venues, with **ResNets** [26]. The default dataset is **ImageNet** [10]; other datasets are explicitly pointed out.

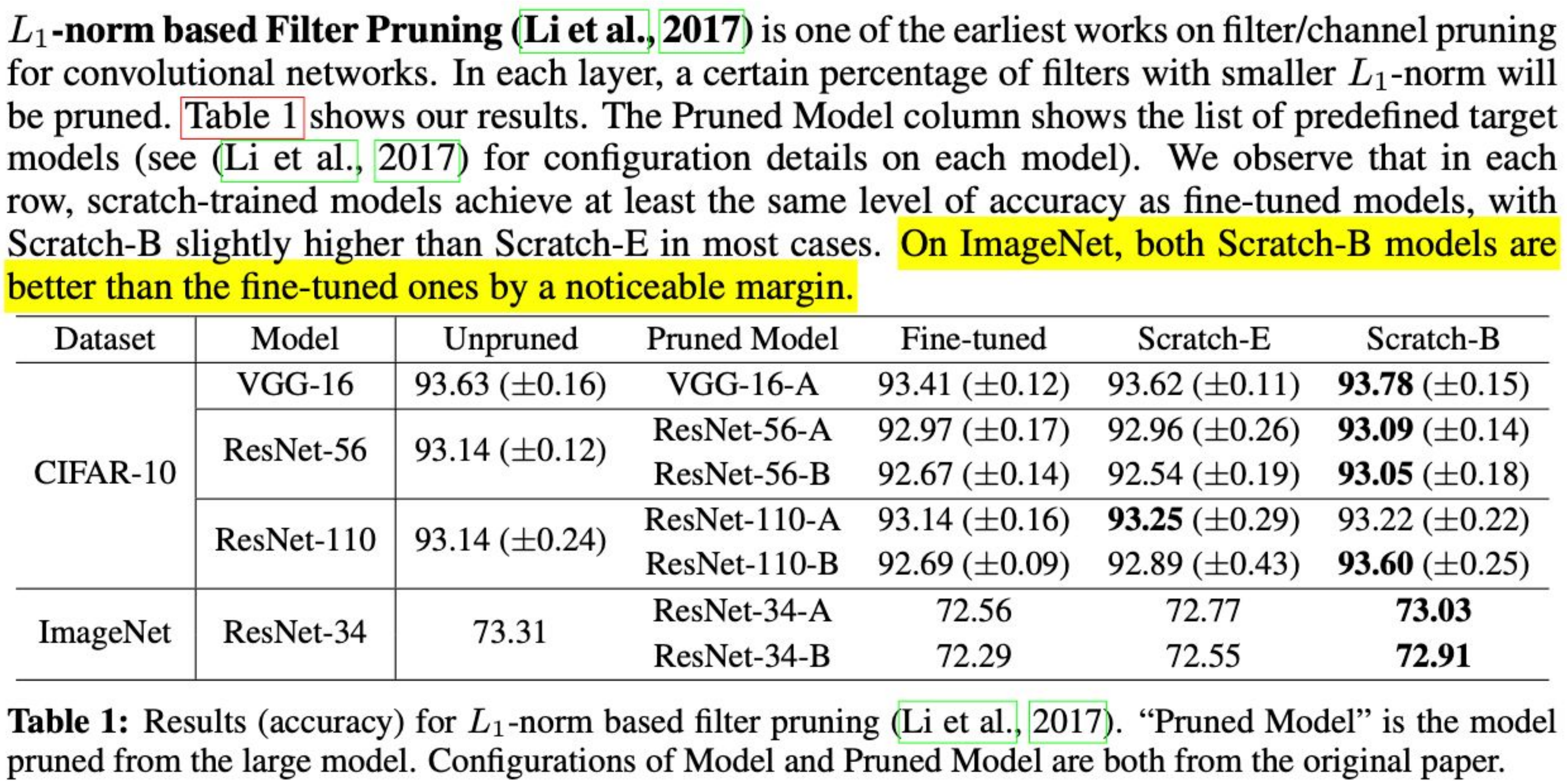
Method	#Epochs	LR schedule
SSL [82]$_{\text{NeurIPS'16}}$ (CIFAR10)	–	0.01
L_1-norm [42]$_{\text{ICLR'17}}$	20	0.001, fixed
DCP [88]$_{\text{NeurIPS'18}}$	60	0.01, step (36/48/54)
GAL-0.5/1 [47]$_{\text{CVPR'19}}$	30	0.01, step decay (10/20)
Taylor-FO [55]$_{\text{CVPR'19}}$	∼25	0.01, step decay (10/20)
Factorized [43]$_{\text{CVPR'19}}$	90	0.01, step decay (30/60)
CCP-AC [60]$_{\text{ICML'19}}$	100	0.001, step decay (30/60/90)
HRank [46] $_{\text{CVPR'20}}$	30×#layers	0.01, step decay (10/20)
GReg-1/2 [79] $_{\text{ICLR'21}}$	90	0.01, step decay (30/60/75)
ResRep [14] $_{\text{ICCV'21}}$	180	0.01, cosine annealing
L_1-norm [42] $_{\text{ICLR'17}}$ **(our reimpl.)**	90	0.01, step decay (30/60/75)


S1: Given the same
Dataset
Network structure
Total sparsity/speedup
S2: S1 + Keep the same base model (M1 arises at this setup)
S1-S4: Four pruning comparison setups
M1-M2: Two mysteries in filter pruning
Sparsifying action
S3: S2 + Keep the same finetuning
Unveil M1: Network trainability is damaged by the sparsifying action, slowing down the finetuning optimization; larger LR makes the model converge faster, thus better performance is observed earlier. The performance is not really "boosted".
Random model (big dense)
Pretraining (big dense)
Base model (big dense)
Pruning method
Pruned model (small dense)
Finetuning (small dense)
Final model (small dense)
Pruning pipeline
M2: Training the pruned model from scratch with adjusted (typically prolonged) epochs can produce the final model with similar performance.
M2 holds under S3, not hold under S2. To unveil M2, we need to unveil M1 first.
M1: A larger finetuning LR can significantly "boost" the final performance.
Scratch training pipeline
Pruned model (small dense)
Training with adjusted epochs (small dense)
Final model (small dense)
S4: S3 + Keep the same pruning cost

Overview of our investigation

Starting from here, we unveil the M2 first: Value of network filter pruning

What made them think so?



L_1**-norm based Filter Pruning** (Li et al., 2017) is one of the earliest works on filter/channel pruning for convolutional networks. In each layer, a certain percentage of filters with smaller L_1-norm will be pruned. Table 1 shows our results. The Pruned Model column shows the list of predefined target models (see (Li et al., 2017) for configuration details on each model). We observe that in each row, scratch-trained models achieve at least the same level of accuracy as fine-tuned models, with Scratch-B slightly higher than Scratch-E in most cases. On ImageNet, both Scratch-B models are better than the fine-tuned ones by a noticeable margin.

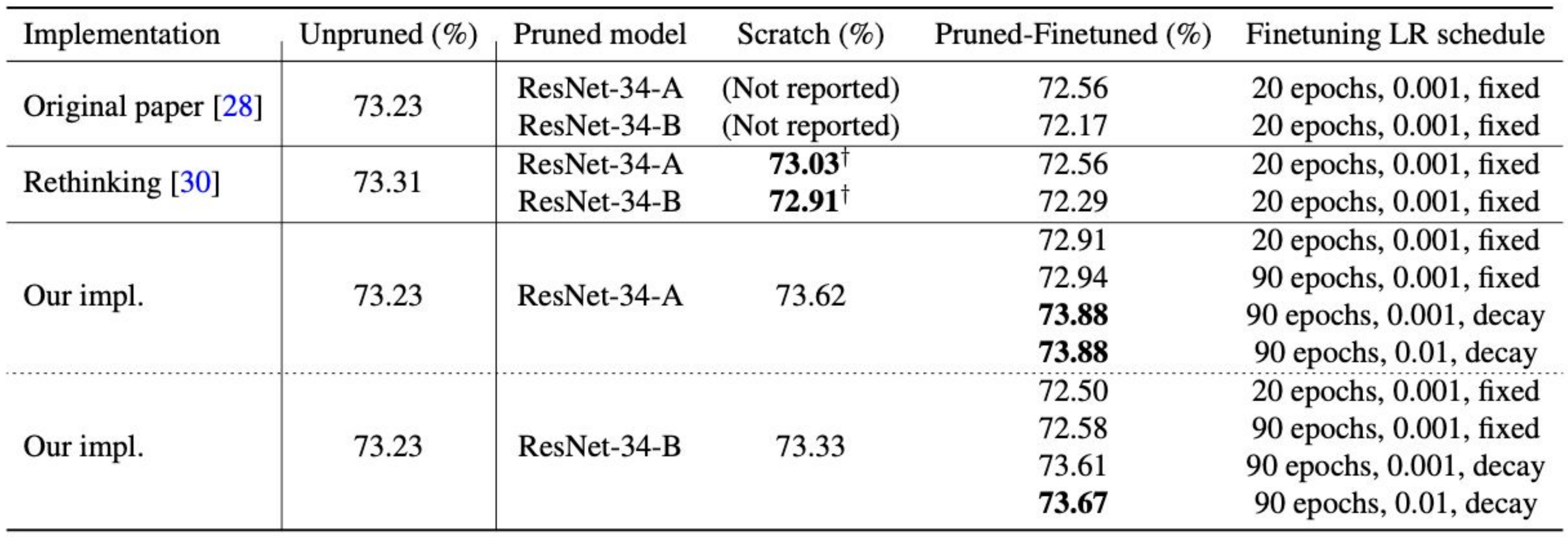
Dataset	Model	Unpruned	Pruned Model	Fine-tuned	Scratch-E	Scratch-B
CIFAR-10	VGG-16	93.63 (±0.16)	VGG-16-A	93.41 (±0.12)	93.62 (±0.11)	**93.78** (±0.15)
	ResNet-56	93.14 (±0.12)	ResNet-56-A	92.97 (±0.17)	92.96 (±0.26)	**93.09** (±0.14)
			ResNet-56-B	92.67 (±0.14)	92.54 (±0.19)	**93.05** (±0.18)
	ResNet-110	93.14 (±0.24)	ResNet-110-A	93.14 (±0.16)	**93.25** (±0.29)	93.22 (±0.22)
			ResNet-110-B	92.69 (±0.09)	92.89 (±0.43)	**93.60** (±0.25)
ImageNet	ResNet-34	73.31	ResNet-34-A	72.56	72.77	**73.03**
			ResNet-34-B	72.29	72.55	**72.91**

Table 1: Results (accuracy) for L_1-norm based filter pruning (Li et al., 2017). "Pruned Model" is the model pruned from the large model. Configurations of Model and Pruned Model are both from the original paper.

[Liu et al., ICLR, 2019]

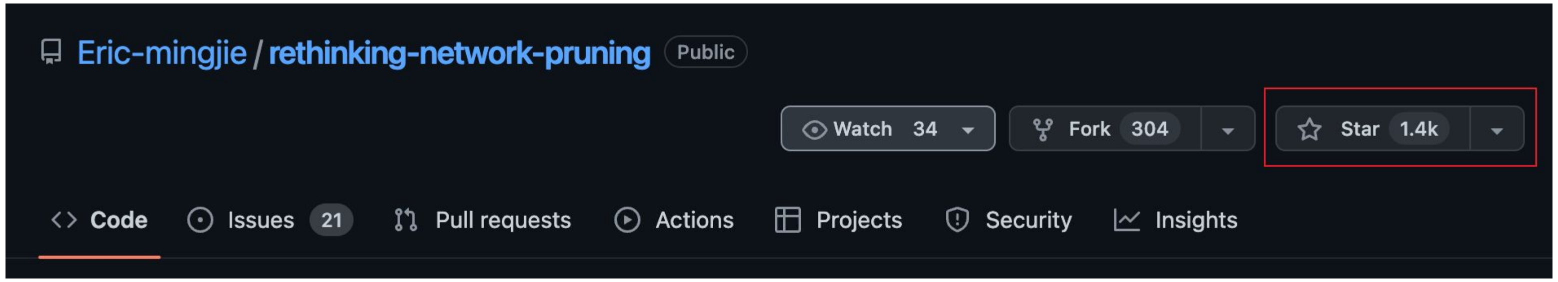
Liu et al. just interpreted their experimental results faithfully.

We reproduced their experiments!



Implementation	Unpruned (%)	Pruned model	Scratch (%)	Pruned-Finetuned (%)	Finetuning LR schedule
Original paper [28]	73.23	ResNet-34-A	(Not reported)	72.56	20 epochs, 0.001, fixed
		ResNet-34-B	(Not reported)	72.17	20 epochs, 0.001, fixed
Rethinking [30]	73.31	ResNet-34-A	**73.03**[†]	72.56	20 epochs, 0.001, fixed
		ResNet-34-B	**72.91**[†]	72.29	20 epochs, 0.001, fixed
Our impl.	73.23	ResNet-34-A	73.62	72.91	20 epochs, 0.001, fixed
				72.94	90 epochs, 0.001, fixed
				73.88	90 epochs, 0.001, decay
				73.88	90 epochs, 0.01, decay
Our impl.	73.23	ResNet-34-B	73.33	72.50	20 epochs, 0.001, fixed
				72.58	90 epochs, 0.001, fixed
				73.61	90 epochs, 0.001, decay
				73.67	90 epochs, 0.01, decay

Table 4. Top-1 accuracy comparison of different implementations of the L_1-norm pruning [28]. Network: **ResNet-34**. Dataset: **ImageNet**. [†]Here we cite the best scratch-training results of [30] (*i.e.*, Scratch-B). We adopt the torchvision models as the unpruned models following common practices. The main point here is that [30] draws the conclusion that scratch training is better than pruning *because of an improper finetuning LR scheme*. With proper finetuning LR schemes ("90 epochs, 0.001, decay" or "90 epochs, 0.01, decay"), pruning is actually *better* than scratch training. Please refer to Sec. 3.4 for detailed discussions.

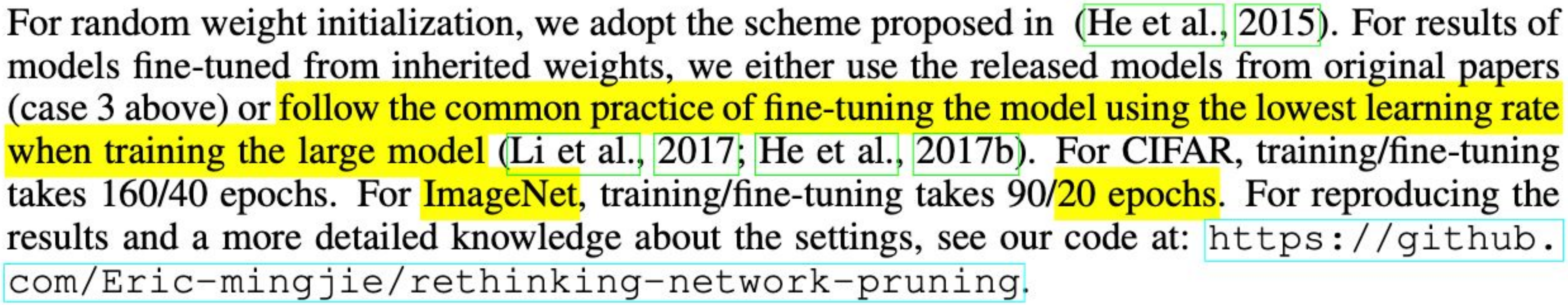


Eric-mingjie / rethinking-network-pruning Public

Watch 34 | Fork 304 | Star 1.4k

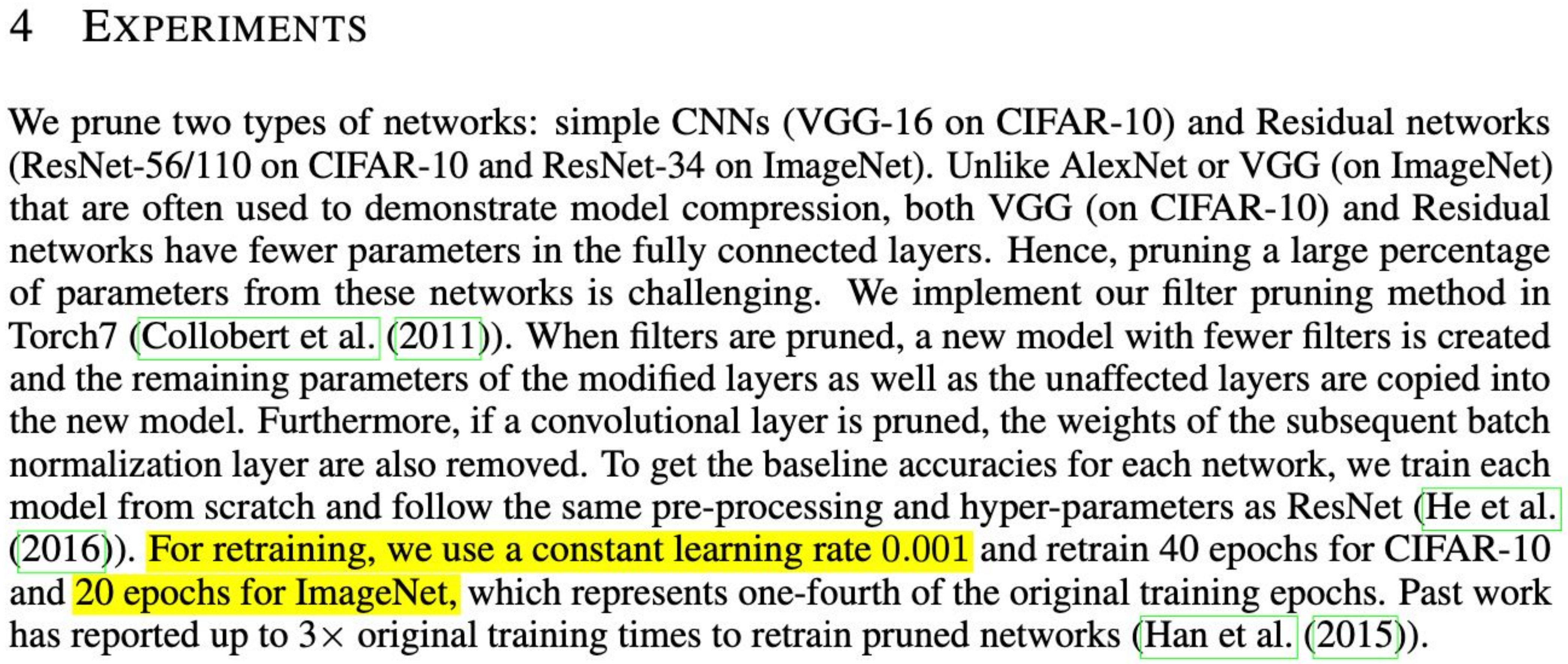
Code | Issues 21 | Pull requests | Actions | Projects | Security | Insights

Devil is in the details


For random weight initialization, we adopt the scheme proposed in (He et al., 2015). For results of models fine-tuned from inherited weights, we either use the released models from original papers (case 3 above) or follow the common practice of fine-tuning the model using the lowest learning rate when training the large model (Li et al., 2017; He et al., 2017b). For CIFAR, training/fine-tuning takes 160/40 epochs. For ImageNet, training/fine-tuning takes 90/20 epochs. For reproducing the results and a more detailed knowledge about the settings, see our code at: https://github.com/Eric-mingjie/rethinking-network-pruning.

[Liu et al., ICLR, 2019]

Devil is in the details



4 EXPERIMENTS

We prune two types of networks: simple CNNs (VGG-16 on CIFAR-10) and Residual networks (ResNet-56/110 on CIFAR-10 and ResNet-34 on ImageNet). Unlike AlexNet or VGG (on ImageNet) that are often used to demonstrate model compression, both VGG (on CIFAR-10) and Residual networks have fewer parameters in the fully connected layers. Hence, pruning a large percentage of parameters from these networks is challenging. We implement our filter pruning method in Torch7 (Collobert et al. (2011)). When filters are pruned, a new model with fewer filters is created and the remaining parameters of the modified layers as well as the unaffected layers are copied into the new model. Furthermore, if a convolutional layer is pruned, the weights of the subsequent batch normalization layer are also removed. To get the baseline accuracies for each network, we train each model from scratch and follow the same pre-processing and hyper-parameters as ResNet (He et al. (2016)). For retraining, we use a constant learning rate 0.001 and retrain 40 epochs for CIFAR-10 and 20 epochs for ImageNet, which represents one-fourth of the original training epochs. Past work has reported up to $3\times$ original training times to retrain pruned networks (Han et al. (2015)).

[Li et al., ICLR, 2017]

Ha, turns out that Li et al. [2017, ICLR] used a (severely) sub-optimal LR schedule, passed on by Liu et al. [2019, ICLR]. So simple!

What would scratch training be, compared to L1-norm pruning using a better fine tuning LR? Will that lead to a *different* conclusion?

One table to show them all

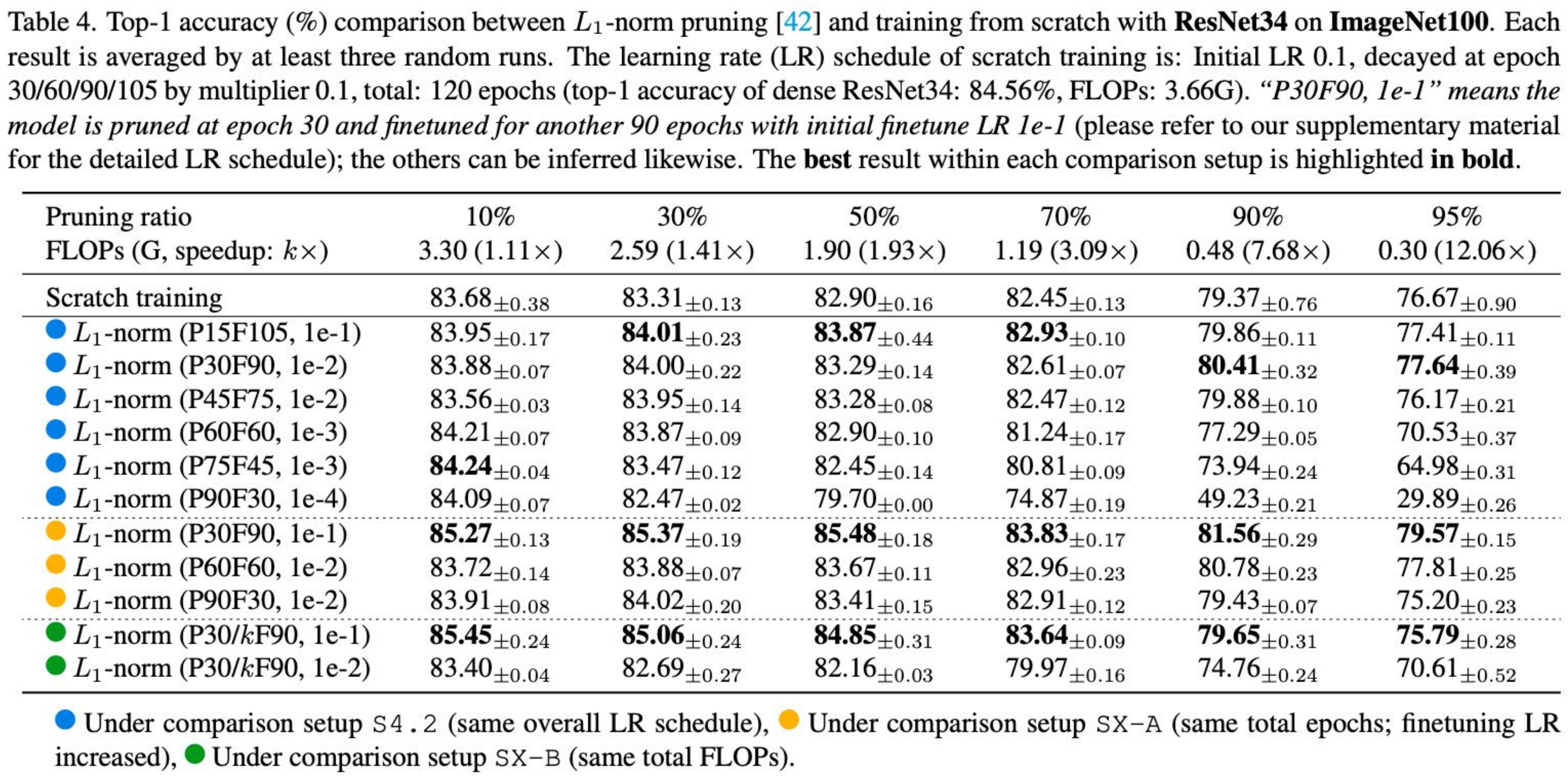


Table 4. Top-1 accuracy (%) comparison between L_1-norm pruning [42] and training from scratch with **ResNet34** on **ImageNet100**. Each result is averaged by at least three random runs. The learning rate (LR) schedule of scratch training is: Initial LR 0.1, decayed at epoch 30/60/90/105 by multiplier 0.1, total: 120 epochs (top-1 accuracy of dense ResNet34: 84.56%, FLOPs: 3.66G). *"P30F90, 1e-1" means the model is pruned at epoch 30 and finetuned for another 90 epochs with initial finetune LR 1e-1* (please refer to our supplementary material for the detailed LR schedule); the others can be inferred likewise. The **best** result within each comparison setup is highlighted **in bold**.

Pruning ratio FLOPs (G, speedup: $k\times$)	10% 3.30 (1.11×)	30% 2.59 (1.41×)	50% 1.90 (1.93×)	70% 1.19 (3.09×)	90% 0.48 (7.68×)	95% 0.30 (12.06×)
Scratch training	$83.68_{\pm 0.38}$	$83.31_{\pm 0.13}$	$82.90_{\pm 0.16}$	$82.45_{\pm 0.13}$	$79.37_{\pm 0.76}$	$76.67_{\pm 0.90}$
● L_1-norm (P15F105, 1e-1)	$83.95_{\pm 0.17}$	$\mathbf{84.01}_{\pm 0.23}$	$\mathbf{83.87}_{\pm 0.44}$	$\mathbf{82.93}_{\pm 0.10}$	$79.86_{\pm 0.11}$	$77.41_{\pm 0.11}$
● L_1-norm (P30F90, 1e-2)	$83.88_{\pm 0.07}$	$84.00_{\pm 0.22}$	$83.29_{\pm 0.14}$	$82.61_{\pm 0.07}$	$\mathbf{80.41}_{\pm 0.32}$	$\mathbf{77.64}_{\pm 0.39}$
● L_1-norm (P45F75, 1e-2)	$83.56_{\pm 0.03}$	$83.95_{\pm 0.14}$	$83.28_{\pm 0.08}$	$82.47_{\pm 0.12}$	$79.88_{\pm 0.10}$	$76.17_{\pm 0.21}$
● L_1-norm (P60F60, 1e-3)	$84.21_{\pm 0.07}$	$83.87_{\pm 0.09}$	$82.90_{\pm 0.10}$	$81.24_{\pm 0.17}$	$77.29_{\pm 0.05}$	$70.53_{\pm 0.37}$
● L_1-norm (P75F45, 1e-3)	$\mathbf{84.24}_{\pm 0.04}$	$83.47_{\pm 0.12}$	$82.45_{\pm 0.14}$	$80.81_{\pm 0.09}$	$73.94_{\pm 0.24}$	$64.98_{\pm 0.31}$
● L_1-norm (P90F30, 1e-4)	$84.09_{\pm 0.07}$	$82.47_{\pm 0.02}$	$79.70_{\pm 0.00}$	$74.87_{\pm 0.19}$	$49.23_{\pm 0.21}$	$29.89_{\pm 0.26}$
● L_1-norm (P30F90, 1e-1)	$\mathbf{85.27}_{\pm 0.13}$	$\mathbf{85.37}_{\pm 0.19}$	$\mathbf{85.48}_{\pm 0.18}$	$\mathbf{83.83}_{\pm 0.17}$	$\mathbf{81.56}_{\pm 0.29}$	$\mathbf{79.57}_{\pm 0.15}$
● L_1-norm (P60F60, 1e-2)	$83.72_{\pm 0.14}$	$83.88_{\pm 0.07}$	$83.67_{\pm 0.11}$	$82.96_{\pm 0.23}$	$80.78_{\pm 0.23}$	$77.81_{\pm 0.25}$
● L_1-norm (P90F30, 1e-2)	$83.91_{\pm 0.08}$	$84.02_{\pm 0.20}$	$83.41_{\pm 0.15}$	$82.91_{\pm 0.12}$	$79.43_{\pm 0.07}$	$75.20_{\pm 0.23}$
● L_1-norm (P30/kF90, 1e-1)	$\mathbf{85.45}_{\pm 0.24}$	$\mathbf{85.06}_{\pm 0.24}$	$\mathbf{84.85}_{\pm 0.31}$	$\mathbf{83.64}_{\pm 0.09}$	$\mathbf{79.65}_{\pm 0.31}$	$\mathbf{75.79}_{\pm 0.28}$
● L_1-norm (P30/kF90, 1e-2)	$83.40_{\pm 0.04}$	$82.69_{\pm 0.27}$	$82.16_{\pm 0.03}$	$79.97_{\pm 0.16}$	$74.76_{\pm 0.24}$	$70.61_{\pm 0.52}$

● Under comparison setup `S4.2` (same overall LR schedule), ● Under comparison setup `SX-A` (same total epochs; finetuning LR increased), ● Under comparison setup `SX-B` (same total FLOPs).

Value of pruning depends on if we are allowed to use a larger fine tuning LR.

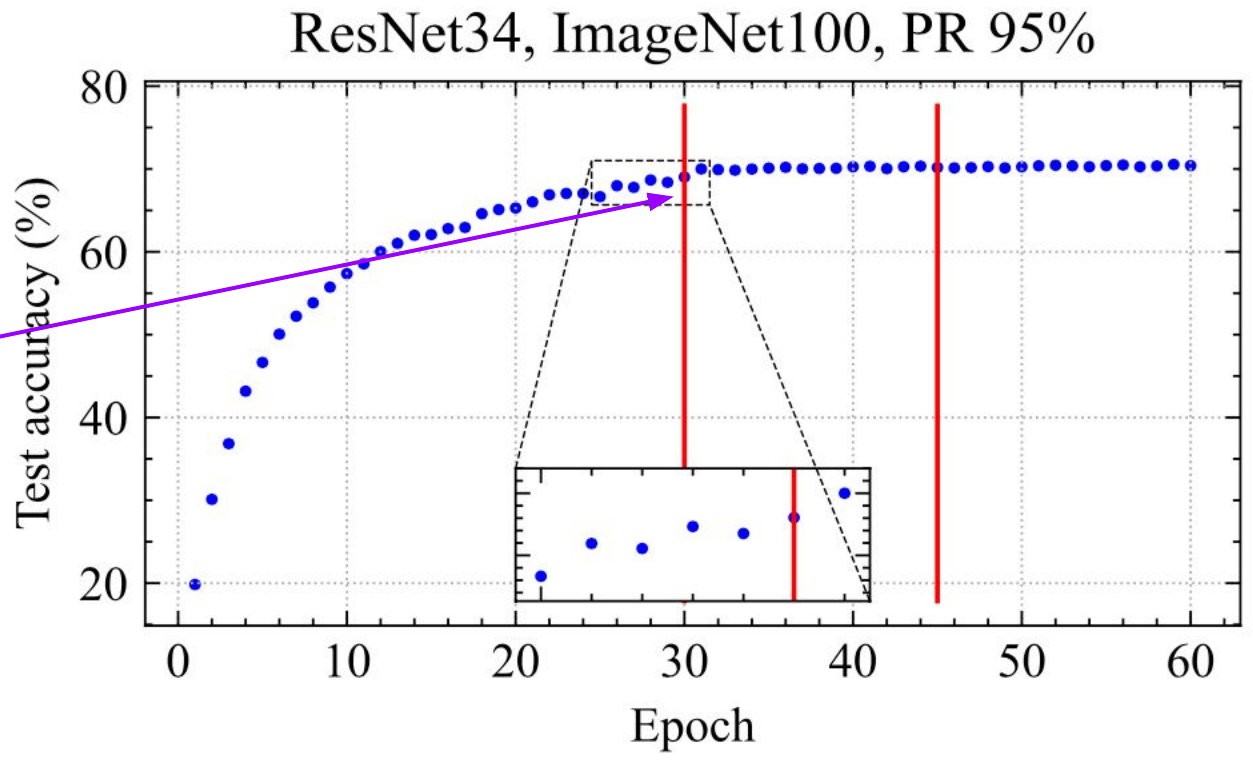
No.	Comparison setups
S1	Compare performance or performance drop on the same dataset and network at the same compression or speedup rate
S2	+Same base model
S3.1	+Same base model
	+Same finetuning epochs
S3.2	+Same base model
	+Same finetuning LR schedule
S4.1	+Same base model
	+Same finetuning LR schedule
	+Same pruning epochs
S4.2	+Same base model
	+Same finetuning LR schedule
	+Same pruning LR schedule
SX-A	+Same epochs of "pretraining + pruning + finetuning"
SX-B	+Same FLOPs of "pretraining + pruning + finetuning"

- ❑ Now we know the value of pruning argument largely depends on the fine-tuning LR setting.
- ❑ If a larger fine tuning LR is allowed to use, their conclusion is wrong. Otherwise, mostly correct.
- ❑ Why? Why a larger fine tuning LR improves the performance significantly?

Starting from here, we unveil the M1: Why a larger fine-tuning LR “improves” the performance so significantly?

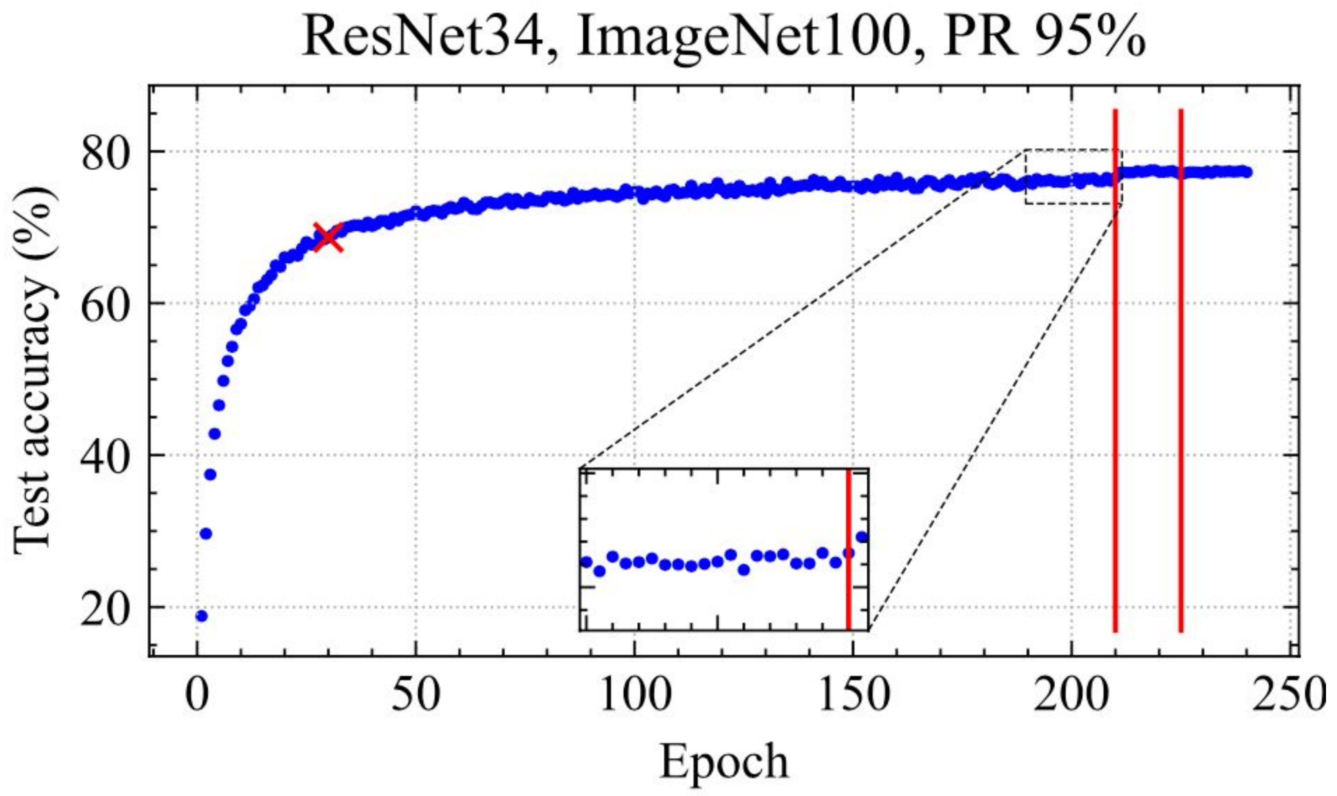
Just plot the learning curve

Is the LR decayed too early?



(a) P60F60, 1e-3

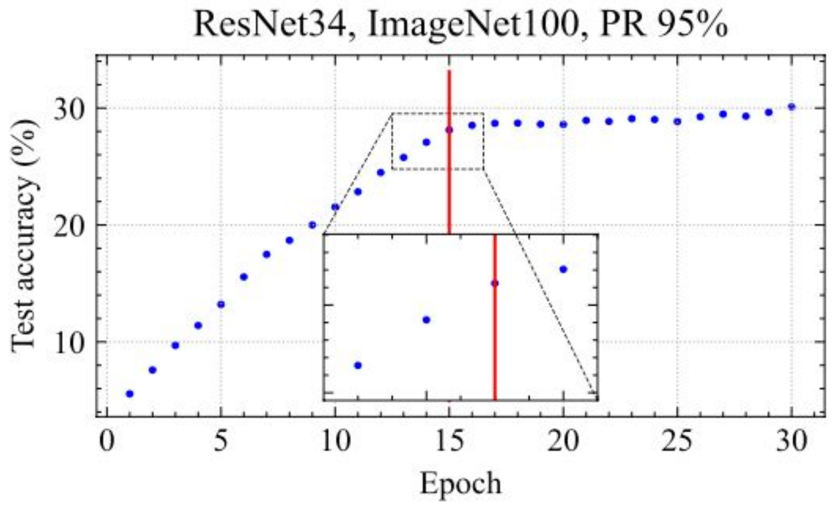
Just plot the learning curve (Cont'd)



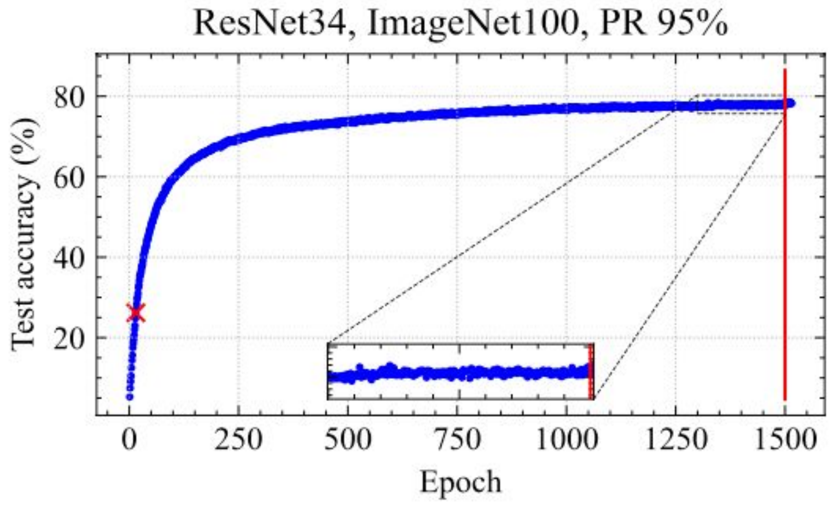
(b) P60F60, 1e-3 (+180 epochs)

Trainability of the pruned model is damaged, slowing down the optimization. The pruned model during finetuning **does not converge at all!**

Just plot the learning curve (Cont'd)



(2.a) P90F30, 1e-4



(2.b) P90F30, 1e-4 (+1485 epochs)

Table 5. Top-1 accuracy (%) comparison of different setups of L_1-norm pruning [42] with **ResNet34** on **ImageNet100**. Pruning ratio: 95%. TA: trainability accuracy (the metric used to measure trainability; see Eq. (1)). This table shows, the performance gap between a smaller LR and a larger LR is not fundamental. It can be closed simply by training more epochs. The root cause that a smaller LR *appears* to under-perform a larger LR is simply that the model trained by the smaller LR does *not* fully converge.

Finetuning setup	Top-1 acc. (%)	TA (%)
P30F90, 1e-1	$79.57_{\pm 0.15}$	88.00
P30F90, 1e-2	$77.64_{\pm 0.39}$	77.45
P30F90, 1e-2 (+30 epochs)	$79.12_{\pm 0.19}$	/
P30F90, 1e-2 (+60 epochs)	$\mathbf{79.59}_{\pm 0.25}$	/
P60F60, 1e-2	$\mathbf{77.81}_{\pm 0.25}$	87.39
P60F60, 1e-3	$70.53_{\pm 0.37}$	68.19
P60F60, 1e-3 (+60 epochs)	$75.71_{\pm 0.09}$	/
P60F60, 1e-3 (+120 epochs)	$77.17_{\pm 0.13}$	/
P60F60, 1e-3 (+180 epochs)	$77.33_{\pm 0.09}$	/
P90F30, 1e-2	$75.20_{\pm 0.23}$	84.83
P90F30, 1e-4	$29.89_{\pm 0.26}$	37.93
P90F30, 1e-4 (+60 epochs)	$60.69_{\pm 0.17}$	/
P90F30, 1e-4 (+270 epochs)	$70.78_{\pm 0.16}$	/
P90F30, 1e-4 (+1485 epochs)	**78.18**	/

What really happens?

- A larger LR does not really "improve" the performance. What really happens is, a larger LR accelerates the optimization process, making the higher performance observed earlier.

- Past pruning works used sub-optimal hyper-parameters, rendering their results *under-estimated.*

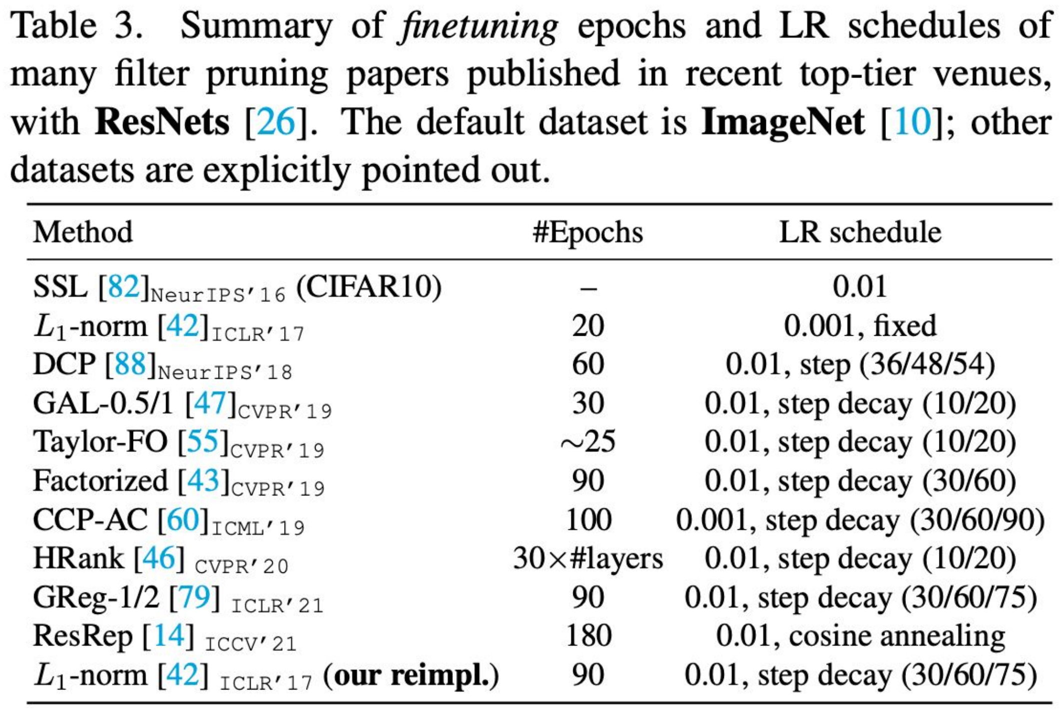
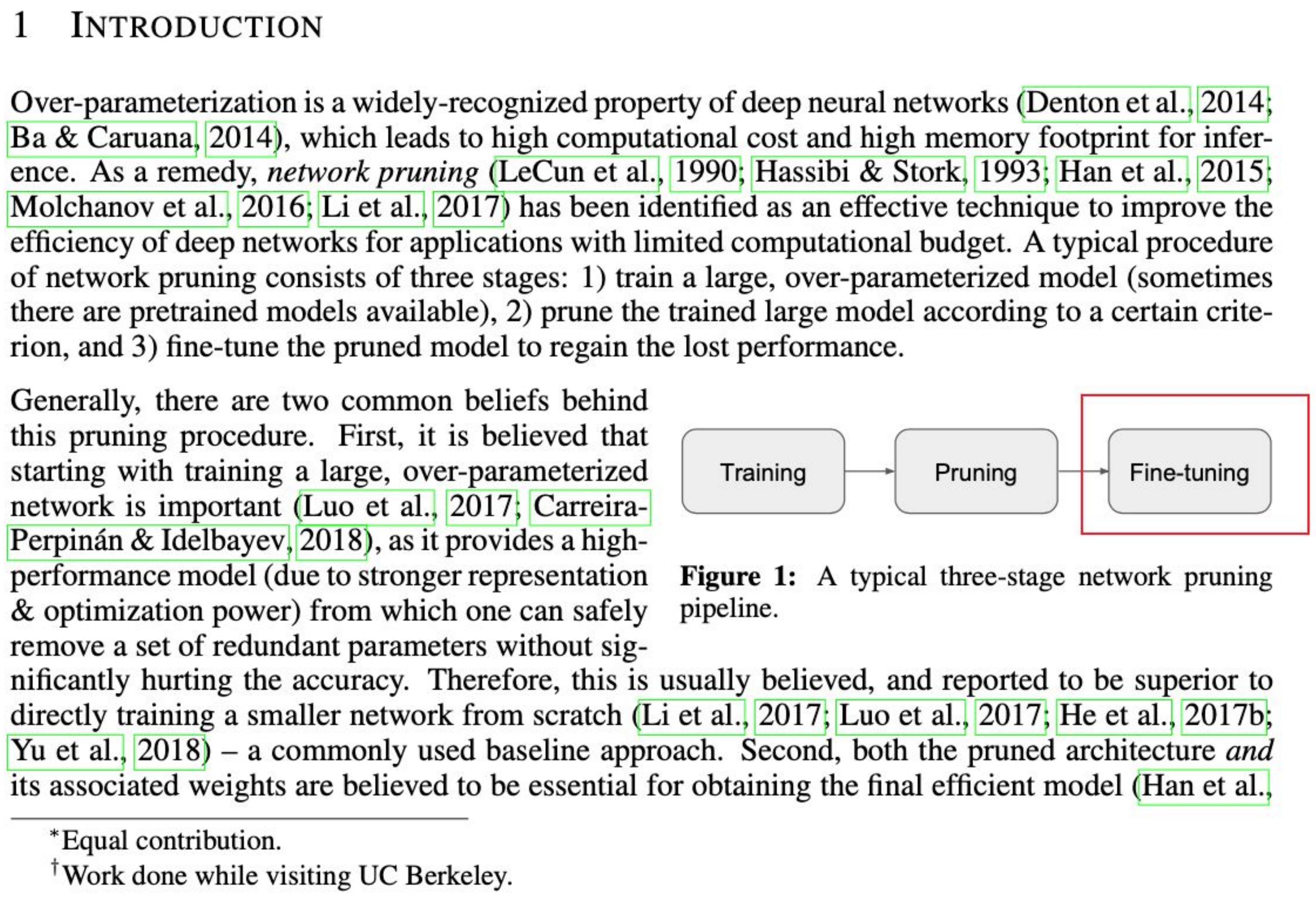


Table 3. Summary of *finetuning* epochs and LR schedules of many filter pruning papers published in recent top-tier venues, with **ResNets** [26]. The default dataset is **ImageNet** [10]; other datasets are explicitly pointed out.

Method	#Epochs	LR schedule
SSL [82]$_{\texttt{NeurIPS'16}}$ (CIFAR10)	–	0.01
L_1-norm [42]$_{\texttt{ICLR'17}}$	20	0.001, fixed
DCP [88]$_{\texttt{NeurIPS'18}}$	60	0.01, step (36/48/54)
GAL-0.5/1 [47]$_{\texttt{CVPR'19}}$	30	0.01, step decay (10/20)
Taylor-FO [55]$_{\texttt{CVPR'19}}$	~25	0.01, step decay (10/20)
Factorized [43]$_{\texttt{CVPR'19}}$	90	0.01, step decay (30/60)
CCP-AC [60]$_{\texttt{ICML'19}}$	100	0.001, step decay (30/60/90)
HRank [46] $_{\texttt{CVPR'20}}$	30×#layers	0.01, step decay (10/20)
GReg-1/2 [79] $_{\texttt{ICLR'21}}$	90	0.01, step decay (30/60/75)
ResRep [14] $_{\texttt{ICCV'21}}$	180	0.01, cosine annealing
L_1-norm [42] $_{\texttt{ICLR'17}}$ **(our reimpl.)**	90	0.01, step decay (30/60/75)



1 INTRODUCTION

Over-parameterization is a widely-recognized property of deep neural networks (Denton et al., 2014; Ba & Caruana, 2014), which leads to high computational cost and high memory footprint for inference. As a remedy, *network pruning* (LeCun et al., 1990; Hassibi & Stork, 1993; Han et al., 2015; Molchanov et al., 2016; Li et al., 2017) has been identified as an effective technique to improve the efficiency of deep networks for applications with limited computational budget. A typical procedure of network pruning consists of three stages: 1) train a large, over-parameterized model (sometimes there are pretrained models available), 2) prune the trained large model according to a certain criterion, and 3) fine-tune the pruned model to regain the lost performance.

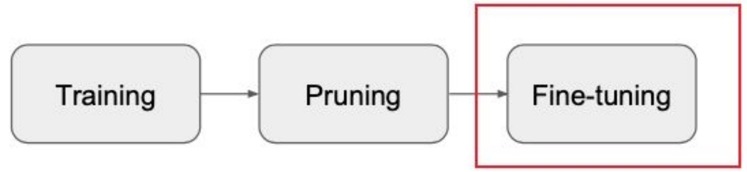


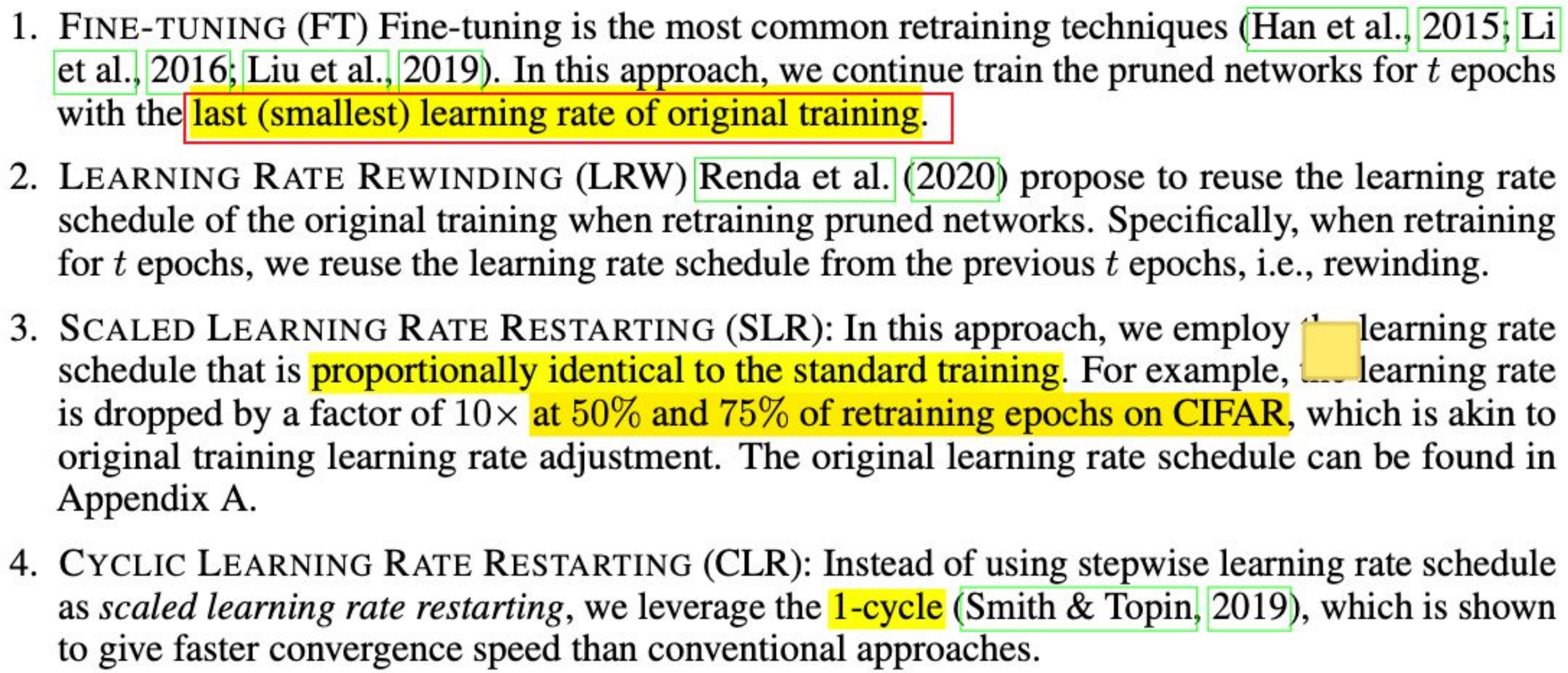
Figure 1: A typical three-stage network pruning pipeline.

Generally, there are two common beliefs behind this pruning procedure. First, it is believed that starting with training a large, over-parameterized network is important (Luo et al., 2017; Carreira-Perpinán & Idelbayev, 2018), as it provides a high-performance model (due to stronger representation & optimization power) from which one can safely remove a set of redundant parameters without significantly hurting the accuracy. Therefore, this is usually believed, and reported to be superior to directly training a smaller network from scratch (Li et al., 2017; Luo et al., 2017; He et al., 2017b; Yu et al., 2018) – a commonly used baseline approach. Second, both the pruned architecture *and* its associated weights are believed to be essential for obtaining the final efficient model (Han et al.,

*Equal contribution.

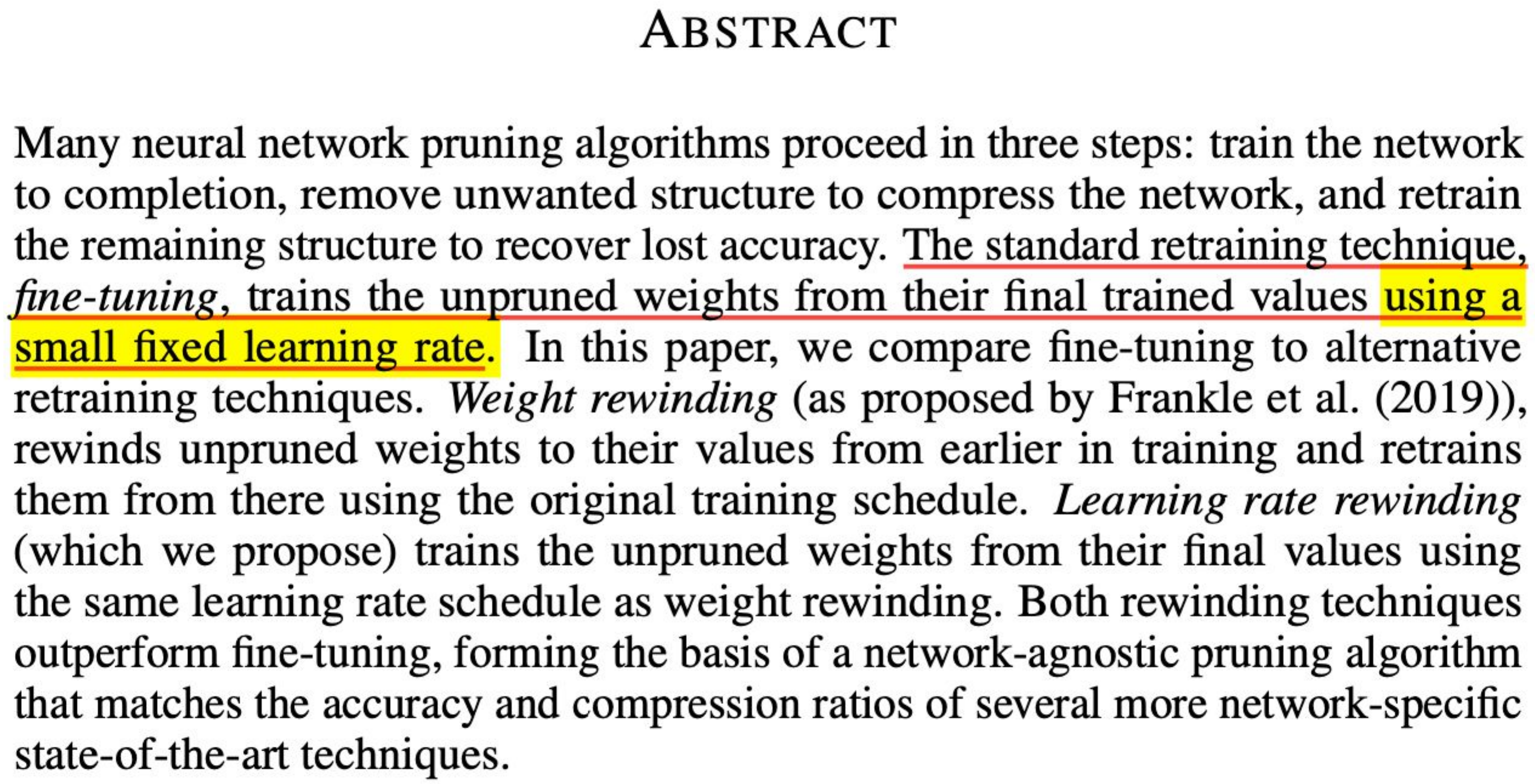
†Work done while visiting UC Berkeley.

[Liu et al., ICLR, 2021]



1. FINE-TUNING (FT) Fine-tuning is the most common retraining techniques (Han et al., 2015; Li et al., 2016; Liu et al., 2019). In this approach, we continue train the pruned networks for t epochs with the last (smallest) learning rate of original training.
2. LEARNING RATE REWINDING (LRW) Renda et al. (2020) propose to reuse the learning rate schedule of the original training when retraining pruned networks. Specifically, when retraining for t epochs, we reuse the learning rate schedule from the previous t epochs, i.e., rewinding.
3. SCALED LEARNING RATE RESTARTING (SLR): In this approach, we employ the learning rate schedule that is proportionally identical to the standard training. For example, the learning rate is dropped by a factor of $10\times$ at 50% and 75% of retraining epochs on CIFAR, which is akin to original training learning rate adjustment. The original learning rate schedule can be found in Appendix A.
4. CYCLIC LEARNING RATE RESTARTING (CLR): Instead of using stepwise learning rate schedule as *scaled learning rate restarting*, we leverage the 1-cycle (Smith & Topin, 2019), which is shown to give faster convergence speed than conventional approaches.

[Le and Hua, ICLR, 2021]



ABSTRACT

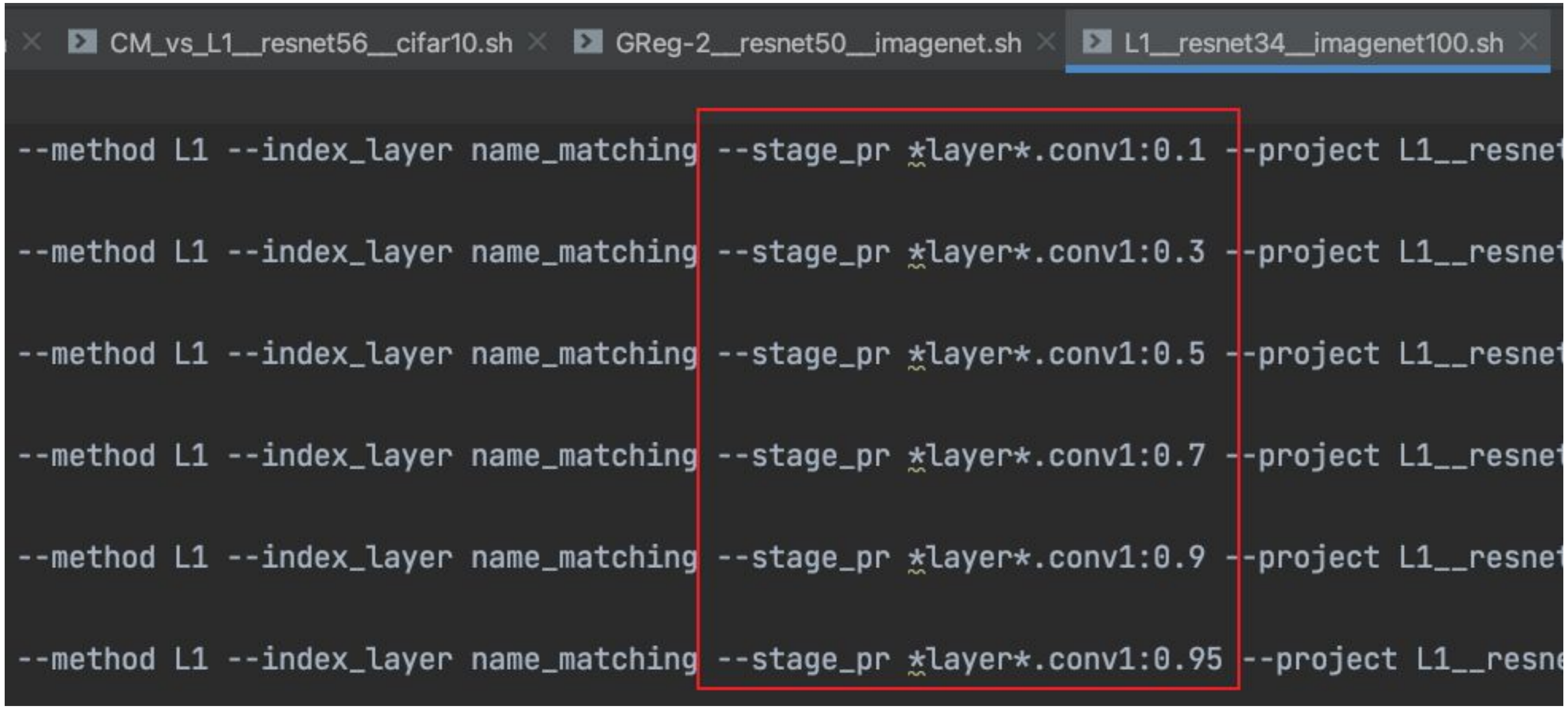
Many neural network pruning algorithms proceed in three steps: train the network to completion, remove unwanted structure to compress the network, and retrain the remaining structure to recover lost accuracy. The standard retraining technique, *fine-tuning*, trains the unpruned weights from their final trained values using a small fixed learning rate. In this paper, we compare fine-tuning to alternative retraining techniques. *Weight rewinding* (as proposed by Frankle et al. (2019)), rewinds unpruned weights to their values from earlier in training and retrains them from there using the original training schedule. *Learning rate rewinding* (which we propose) trains the unpruned weights from their final values using the same learning rate schedule as weight rewinding. Both rewinding techniques outperform fine-tuning, forming the basis of a network-agnostic pruning algorithm that matches the accuracy and compression ratios of several more network-specific state-of-the-art techniques.

[Renda, ICLR, 2021]

Turns out we never had an agreement about even the most *basic* concepts!

Why the trap is so covert?

- Using the same setting for different pruning ratios? **Not very correct**
- At different sparsity levels, the trainability is damaged to different degrees. The hyper-parameters (esp., fine tuning LR and epochs) should be adjusted accordingly.


CM_vs_L1__resnet56__cifar10.sh
GReg-2__resnet50__imagenet.sh
L1__resnet34__imagenet100.sh

```
--method L1 --index_layer name_matching --stage_pr *layer*.conv1:0.1 --project L1__resnet
--method L1 --index_layer name_matching --stage_pr *layer*.conv1:0.3 --project L1__resnet
--method L1 --index_layer name_matching --stage_pr *layer*.conv1:0.5 --project L1__resnet
--method L1 --index_layer name_matching --stage_pr *layer*.conv1:0.7 --project L1__resnet
--method L1 --index_layer name_matching --stage_pr *layer*.conv1:0.9 --project L1__resnet
--method L1 --index_layer name_matching --stage_pr *layer*.conv1:0.95 --project L1__resne
```

Conclusion

Summary of this paper:

- We are motivated by the two mysteries in network (filter) pruning.
- Sort out the comparison setups for later experimenting under a strict control.
- Unveil the two mysteries: M2 → M1
- Trainability is the key to unveil M1.

Takeaways:

- *Why is the state of neural network pruning so confusing?* (1) Non-standard comparison setups (2) Unawareness of the role of trainability.
- Which comparison setup should I choose? **>=S3.2.**
- Reporting all finetuning details are necessary and should be standardized!
- The observation that a larger fine tuning LR “improves” pruning performance is largely a misinterpretation.
- L1-norm pruning is pretty strong!
- The term finetuning is not suitable.

Thanks! Questions?
Code:
https://github.com/mingsun-tse/why-the-state-of-pruning-so-confusing